ADA USER JOURNAL

Volume 33
Number 3
September 2012

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

Further details on our approach to these are given below.

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

News and Product Announcements

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

Reprinted Articles

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

Editorial

The Ada User Journal is produced by a group of hard working volunteers, which dedicate some of their time to guarantee that the Journal continues its quality production in a (as much as possible) timely manner, so it is important, and pleasant, when we have the opportunity to enlarge this group with new and motivated helping hands. It is thus with contentment that I inform that starting with this issue, Patricia López Martínez, from the University of Cantabria, Spain, takes the role of Assistant Editor in the Editorial Board of the Journal. Patrícia is, since last August, assisting in the production of the Journal, successfully and smoothly taking much of the production tasks. I take the opportunity to welcome her and wish her a very successful term.

And the term started with really an impressive issue. These 94 pages (the largest issue in my term as Editor-in-Chief of the Journal) provide a very interesting and diversified set of contents, which, I am sure, will draw the attention and please the reader.

First we are glad to publish the announcement, made at the Ada-Europe 2012 conference, of the winners of the Ada Way contest, a team from the University of Padua, Italy, formed by Ricardo Aguirre Reyes, Andrea Graziano, Marco Teoli, and Alberto Zuccato. We are glad that these students have also accepted our invitation, and provided to the Journal a report on the main design decisions and difficulties found in developing the simulator. The contest enters now a "try and beat me" challenge, where any team can attempt to improve over this reference implementation.

We also continue the publication of the Ada 2012 Rationale, by John Barnes, with the chapter describing the changes in the tasking and real-time domain, of which possibly the most relevant is the mechanism to control the allocation of tasks on multiprocessors. The emergence and challenges of these architectures require that languages consider new ways to control and schedule tasks, in order to fully utilize the available parallel power. The Ada concurrency model has always been an important mark of the language and it is thus important that Ada continues to be in the forefront of concurrency, and now parallelism, support. Ada 2012 already incorporates some of these features, but more work is undoubtedly still necessary. And, although a coincidence, it is a pleasure that I note that this issue provides the announcement and call for papers for the next IRTAW; the 16th International Real-Time Ada Workshop will take place April 2013 in the beautiful scenery of York, in the UK. IRTAW is the main forum dedicated to the advances of Ada in the tasking and real-time domain and has provided many of the advances in this area in the language revisions.

The issue also provides the position papers and summary for the two panels that took place in the Ada-Europe 2012 conference. The first of these, entitled "What is Language Technology in Our Time?", addressed the increasingly important issue of a language needing an eco-system of frameworks and tools around it in order to reach out to prospective users. This panel was moderated by Tullio Vardanega, from the University of Padua, Italy, and included as panelists Bertrand Meyer, Eiffel Software, Switzerland; Franco Gasperoni, AdaCore, France; Erhard Plödereder, University of Stuttgart, Germany; and José María Martínez, Cassidian, Spain. This issue includes the position papers of moderator and panelists, as well as a summary of the panel discussion by Albert Llemosí, Universitat de les Illes Balears, Spain.

The second panel, entitled "Reliable Software, a Perspective from Industry", discussed the industrial perspective on the software technology for reliable software. It was moderated by Jørgen Bundgaard, Ada in Denmark, and included as panelists Ana Rodríguez, GMV, Spain; Steen Palm, Terma A/S, Denmark and Rick Sward, MITRE, USA. The issue provides their position papers as well as the summary of the panel by José Ruiz, AdaCore, France.

Continuing with Ada-Europe 2013, we also publish a paper derived from its industrial track, from Frank Dordowsky (ESG, Germany) and Richard Bridges and Holger Tschöpe (Eurocopter, Germany), presenting the design of the I/O data interface of the NH90 software, using Ada generics.

As usual, the issue also provides the News Digest and Calendar sections, by Jacob Sparre Andersen and Dirk Craeynest, their respective editors. The Forthcoming Events section provides, apart from the IRTAW announcement, the advance program of the SIGAda High Integrity Language Technology (HILT 2012) conference and the call for papers for the 18th International Conference on Reliable Software Technologies – Ada-Europe 2013. To finalize, the Ada Gems section provides two gems on Iterators in Ada 2012 by Emmanuel Briot, from AdaCore.

Luís Miguel Pinho
Porto
September 2012
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation. Email: jacob@jacob-sparre.dk

Contents

Ada-related Organizations

Ada 2012 Language Standard Submitted to ISO

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Mon, 11 Jun 2012 23:59:02 +0000
Subject: Press Release - Ada 2012 Language Standard Submitted to ISO
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

Ada 2012 Language Standard Submitted to ISO Language revision adds contract-based programming, multicore support, and other advanced features

STOCKHOLM, SWEDEN, June 12, 2012 - At the Ada-Europe 2012 conference in Stockholm, the Ada Resource Association (ARA) and Ada-Europe today announced the completion of the design of the latest version of the Ada programming language and the submission of the reference manual to the International Organization for Standardization (ISO) for approval. The language revision, known as Ada 2012, is under the auspices of ISO/IEC JTC1/SC22/WG9 and was conducted by the Ada Rapporteur Group (ARG) subunit of WG9, with sponsorship in part from the ARA and Ada-Europe.

Ada 2012 brings significant enhancements to Ada, most notably in the area of "contract-based programming." New features here include the ability to specify preconditions and postconditions for subprograms, and invariants for private (encapsulated) types. These take the form of Boolean expressions that can be interpreted (under programmer control) as run-time conditions to be checked. The contract-based programming features fit in smoothly with Ada's Object-Oriented Programming model, and support the type substitutability guidance supplied in the Object-Oriented Technologies and Related Techniques Supplement (DO-332) to the new avionics software safety standard DO-178C / ED-12C.

Other new features in Ada 2012 include enhancements to the containers library, additional expressiveness through features such as conditional expressions and more powerful iterators, and support for multicore platforms (task affinities, and the extension of the Ravenscar profile - standardized in Ada 2005 as an efficient and predictable tasking subset for high-integrity real-time systems - to multiprocessor and multicore environments).

"Ada 2012 is a major advance in the state of the art," said Dr. Edmond Schonberg, Rapporteur of the ARG. "The new features answer real user needs, and help cement Ada's reputation as a language of choice for systems where reliability, safety, and security are needed."

"The Ada Rapporteur Group did an excellent job of carrying out the language revision," said Dr. Joyce Tokar, Convenor of WG9.

"Special thanks to Randy Brukardt for his editorial work on the Language Reference Manual, and to Ed Schonberg and all the other ARG members. Ada 2012 is a significant technical accomplishment."

Formal ISO approval of the Ada 2012 revision is expected in late 2012.

With the growing complexity of software systems in most aspects of our daily professional and personal life, program correctness is a paramount concern. Ada 2012 provides outstanding solutions to that end, which can be applied both in industry for production software development, and in academia for teaching and research.

Ada-Europe Announces 1st Ada Way Award Winners

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Mon, 30 Jul 2012 21:03:06 +0000
Subject: Ada-Europe Announces 1st Ada Way Award Winners and Try-and-Beat-Me Challenge
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

Ada-Europe Announces First "Ada Way" Award Winners and "Try and Beat Me" Challenge.

Brussels, Belgium (July 30, 2012) - Ada-Europe, www.ada-europe.org, the international organization that promotes the knowledge and use of the Ada programming language in European academia, research and industry, launched "The Ada Way" annual student programming contest in September 2010. The first challenge was to build a software simulator of a football (soccer) match. The submitted code had to include a software core implementing the logic of the simulation, and read-write graphical panels for interactive team management.

The evaluation committee chose one of the submissions made until April 2012, which at the recent Ada-Europe 2012 conference in Stockholm was proclaimed the reference implementation. The winning student team, formed by Ricardo Aguirre Reyes, Andrea Graziano, Marco Teoli, and Alberto Zuccato, received a laminated Ada Way Award donated by Ada-Europe to commend the outstanding quality of their submission.

In evaluating the authors' submission the evaluation committee reported: "This implementation of the Ada Way Soccer Simulation reveals extraordinary care and engineering skill, and represents a working, scalable, well-documented, and well-structured solution. From reading the technical documentation, it is clear that the development team faced many challenges, and in every case determined an appropriate solution through a combination of thoughtful analysis, experimentation, and clever design." The story of their implementation will be told in a forthcoming issue of the Ada User Journal, the quarterly magazine of Ada-Europe. In due course, the winning team will receive all elements of the prize attached to their fine achievement.

Today, Ada-Europe is pleased to announce that the full source of the reference implementation is posted on the Ada Way page, www.ada-europe.org/AdaWay, along with its accompanying technical specification, user manual and build instructions, a short demo video clip and an image of the award.

The reference implementation is now proposed for a "Try and Beat Me" open-ended challenge: any student team willing to take that challenge is invited to make a submission that attempts to improve over the reference implementation under any of the evaluation criteria listed on the Ada Way page. On 15 May of every year, any such new submission will be evaluated and the best one will be awarded a minor

prize and will replace the previous reference submission in the continuation of the try-and-beat-me challenge.

The evaluation will be performed by a team of distinguished Ada experts comprised of: John Barnes (author of the famous Programming in Ada books), S. Tucker Taft (leader of the Ada 95 language revision), Pascal Leroy (leader of the Ada 2005 language revision), Ed Schonberg(co-author of the open-source GNAT Ada compiler and toolset), Joyce Tokar (convenor of the ISO working group on the Ada language standards), etc.

The winning team will be announced at the Ada-Europe yearly conference subsequent to the cut-off date at which submissions entered the challenge. The prize for this challenge includes a framed award, an Ada book of choice, visibility in electronic and printed media, one free registration and a monetary grant of up to EUR 1000 for the winning team to use for collective participation at any future Ada-Europe conference of choice within two calendar years after selection for the prize.

Ada-Europe wants the competition to be fun and instructive. The implementation does not need to be 100% Ada, but the essence must of course be. Tullio Vardanega, president of Ada-Europe, stated: "The winning submission must be a reference for good Ada programming, software design, and innovation."

For all details, please refer to the official web page of “The Ada Way”, www.ada-europe.org/AdaWay.

Ada-related Events

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

Open Ada-DK Meetings

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 02 Jul 2012 10:56:49 +0200
Subject: The July 2012 Open Ada-DK Meeting
Newsgroups: ada-dk.misc

July 3rd 2012 from 1730 -> ? marks the day and time when the twentieth open Ada-DK meeting is being held.

The “open” part means that the meeting is not a members-only affair, but that anybody interested in Ada is welcome, so feel free to invite whomever you might believe could be interested in spending an evening talking about Ada programming.

Participation in the meeting is of course free.

[Also open Ada-DK meetings August 7th and September 4th. —sparre]

Public Ada Courses in Carlsbad, CA

From: Ed Colbert <colbert@abssw.com>
Date: Mon, 16 Jul 2012 19:04:37 -0700
Subject: [Announcing] Public Ada Courses 20-24 August 2012 in Carlsbad CA
Newsgroups: comp.lang.ada

Absolute Software will be holding a public Ada course during the week of 20 August 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.

[also in 1-5 October 2012 —sparre]

Ada-related Resources

Clean out those dead links

From: wrp <i3text@gmail.com>
Date: Sat, 9 Jun 2012 15:11:11 -0700
Subject: Clean out those dead links
Newsgroups: comp.lang.ada

I'm currently searching online for Ada resources. What impresses me most so far is the number of dead links I find.

When I find a page listing Ada resources, I'm finding that usually about 80% of the links are dead. It creates a pretty bad impression to see that so many projects have been abandoned. What's even worse, though, is to see that people who once cared enough about Ada to promote it on their web site now don't think about it, and probably haven't for several years.

Don't you agree that can give people a really bad impression of the state of Ada?

So, if you have an Ada page, why don't you spend a few minutes to clean it up?

From: Patrick <patrick@spellingbeewinnars.org>
Date: Sat, 9 Jun 2012 15:36:40 -0700
Subject: Re: Clean out those dead links
Newsgroups: comp.lang.ada

As someone new to Ada. I have found this very discouraging at the beginning. I only gained confidence to make an investment in the language once I realized it would play nice with C and that compiler support was likely to continue for 10-20 more years.

Even Adacore's site has a lot of dead links. The trouble is how can we the people who care, get those who don't to update their sites? Easier said than done.

From: Nasser M. Abbasi <nma@12000.org>
Date: Sat, 09 Jun 2012 18:47:16 -0500
Subject: Re: Clean out those dead links
Newsgroups: comp.lang.ada

“So, if you have an Ada page, why don't you spend a few minutes to clean it up?”

Agree. But it does really take more than few minutes. When I cleaned the links on my Ada links page: http://12000.org/my_notes/ada/original_web_page.htm

I record the time that the link was check to be valid, and if the link is broken, search the net for where the link gone (this takes time) so that to update it, and there is no alternative link to be found, then remove it.

I remember it took me few long hrs to do the whole page.

I guess many are busy with work and family, not everyone has time. But your point is valid. I have the same experience. This reminds me, I need to go check my Ada links now. I see I last validated them in 2005.

:)

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 10 Jun 2012 11:22:00 -0400
Subject: Re: Clean out those dead links
Newsgroups: comp.lang.ada

webcheck can help here (http://www.stephe-leake.org/ada/webcheck.html).

It scans a web site, checking the validity of each link, and outputs a list of the broken ones.

That automates the first step, at least.

DIY Operating system in Ada

From: Ada in Denmark
Date: Mon, 25 Jun 2012 09:26:39 +0000
Subject: DIY Operating system using in Ada
URL: http://ada-dk.org/2012/06/diy-operating-system-using-in-ada/

Ada in Denmark: DIY Operating system using in Ada:

The Bare bones tutorial over at OSDev [1] has been ported to Ada!

It supports x86 targets ATM, but will – according to the author – be extended to ARM (with the Raspberry Pi in mind).

The tutorial can be found at [2] and the source test is located at GitHub [3].

[1] http://wiki.osdev.org/Bare_Bones

[2] http://wiki.osdev.org/Ada_Bare_bones

[3] https://github.com/Lucretia/bare_bones

Ada 2012 Rationale

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 2 Jul 2012 19:12:34 -0500
Subject: New Ada 2012 Rationale Edition available
Newsgroups: comp.lang.ada

A new edition of the Ada 2012 Rationale is available at:

http://www.ada-auth.org/standards/rationale12.html

This edition of the Rationale combines the first three chapters of the Rationale into a single document, fixes a number of errors, adds an index, and adds discussion of various details of Ada 2012 that were changed since the original publication of these chapters in the Ada User Journal. We expect that additional chapters will be added to this edition about every three months.

The Rationale for Ada 2012 provides an overview of new Ada 2012 features, examples of their use, compatibility with Ada 95 and 2005, and more. It was written by John Barnes, and was sponsored in part by the Ada Resource Association. This is an unofficial description of the language; refer to the proposed Ada 2012 standard for detailed language rules.

Ada-related Tools

Fuzzy machine learning framework

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 28 May 2012 12:08:38 +0200
Subject: ANN: Fuzzy machine learning framework v1.2
Newsgroups: comp.lang.ada

The software is a library as well as a GTK GUI front-end for machine learning projects. Features:

- Based on intuitionistic fuzzy sets and the possibility theory;
- Features are fuzzy;
- Fuzzy classes, which may intersect and can be treated as features;
- Numeric, enumeration features and ones based on linguistic variables;
- Derived and evaluated features;
- Classifiers as features for building hierarchical systems;
- User-defined features;
- An automatic classification refinement in case of dependent features;
- Incremental learning;
- Object-oriented software design;
- Features, training sets and classifiers are extensible objects;
- Automatic garbage collection;
- Generic data base support (through ODBC);
- Text I/O and HTML routines for features, training sets and classifiers;
- GTK+ widgets for features, training sets and classifiers;
- Examples of use.

This release is packaged for Windows, Fedora (yum) and Debian (apt). The software is public domain (licensed under GM GPL).

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

Turbo Pascal 7 library port

From: Blady <p.p11@orange.fr>
Date: Sun, 10 Jun 2012 01:55:47 -0700
Subject: [ANN] TP7 emulation V2.6 with GTK-Ada
Newsgroups: comp.lang.ada

The Turbo Pascal 7 library port in Ada is intended to assist when porting Turbo Pascal programs to Ada. It can be combined with P2Ada translator [1]. The implementation is based on GtkAda.

It can be used as a basic multi-purpose library for simple graphic stuff. Basic but quite complete and easy to use as the original library was ;-) By the way, it provides an embedded text console.

With only few lines you can operate a full text terminal:

```
with TP7.System;
with TP7.Crt; -- if you comment this line then
              -- I/O use stdinout;
procedure Hello_GtkAda is
  use TP7, TP7.System;
  N : Byte;
begin
  Write ("How many hello ? ");
  Readln (N);
  for I in 1 .. N loop
    Writeln ("Hello with GtkAda console.");
  end loop;
end Hello_GtkAda;
```

See screenshot at [2].

Complete source code at [3].

[1] http://sourceforge.net/projects/p2ada/

[2] http://blady.pagesperso-orange.fr/telechargements/tp-ada/tp7ada-mini.png

[3] http://p2ada.svn.sourceforge.net/viewvc/p2ada/extras/tp7ada/current/

QtAda

From: Vadim Godunko <vgodunko@gmail.com>
Date: Sat, 23 Jun 2012 06:52:12 -0700
Subject: Announce: QtAda 3.2.0 preview
Newsgroups: comp.lang.ada

We are pleased to announce preview of next major version of QtAda 3.2.0.

It includes new GPS integration plugin, extended support of Qt's classes, as well as bug fixes. Source code and binary packages for Microsoft Windows can be downloaded from QtAda site:

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

QtAda is an Ada 2005 bindings to the Qt framework's libraries. It allows to create powerful cross-platform GUI applications with native look-and-feel for UNIX/Linux, Microsoft Windows, Mac OS X and embedded devices.

ZanyBlue

From: Michael Rohan <michael@zanyblue.com>
Date: Sun, 1 Jul 2012 20:42:07 -0700
Subject: ANN: ZanyBlue v1.1.0 Beta Available
Newsgroups: comp.lang.ada

A new release of ZanyBlue is now available: 1.1.0 Beta. This is an Ada library currently targeting localization support for Ada (along the lines of Java properties) with supporting message formatting and built-in localization for about 20 locales. The properties files are compiled into Ada sources built with your application and use to access application messages at run-time. The run-time locale is used to select localized messages, if they are available.

Please see the project page on Source Forge for download links, documentation, etc, http://zanyblue.sourceforge.net

This project is licensed under a simple BSD style license.

Math Extensions

From: Simon Wright <simon@pushface.org>
Date: Thu, 12 Jul 2012 20:44:46 +0100
Subject: ANN: Ada 2005 Math Extensions 20120712
Newsgroups: comp.lang.ada

I'm pleased to announce the 20120712 release of the Ada 2005 Math Extensions.

Changes in this release:

The GNAT Project file is now in the top directory of the distribution.

The tests expect AUnit 3 to be installed.

Different releases of LAPACK may alter the sign of eigenvectors returned by the generalized eigensystem code (remember that the generalized eigensystem is Av = lBv, where l is an eigenvalue and v is the corresponding eigenvector). This only affected the tests.

Testing on Debian 6 required an increase in the test limit for complex general eigenvalues tests for Float.

In GNAT GPL 2012 and GCC 4.7, LAPACK and BLAS are no longer used, and therefore aren't provided as part of GNAT on platforms where they aren't natively available. This package requires LAPACK and BLAS to be installed (it links with "-llapack -lblas").

Download:
https://sourceforge.net/projects/gnat-math-extn/files/20120712/

Matreshka

From: Vadim Godunko <vgodunko@gmail.com>
Date: Fri, 13 Jul 2012 12:50:37 -0700
Subject: Announce: Matreshka 0.3.0
Newsgroups: comp.lang.ada

We are pleased to announce availability of new major release of Matreshka 0.3.0. It includes:

- Firebird/Interbase driver for SQL module;
- extensions for AMF module to process UML Testing Profile, OCL and MOF Extensions;
- text codecs for ISO-8859-5 and ASCII character encodings;
- API improvement, bug fixes and performance improvements;
- GNAT GPL 2012 support.

Matreshka is framework for development of information systems in Ada. It provides:

- localization, internationalization and globalization support;
- XML processor;
- FastCGI support;
- SQL database access;
- UML processing module.

http://forge.ada-ru.org/matreshka/wiki

From: Patrick <patrick@spellingbeewinnars.org>
Date: Sun, 15 Jul 2012 22:35:42 -0700
Subject: Non-GPL xml library ?
Newsgroups: comp.lang.ada

Is there an XML library that is not GPL?

From: Simon Wright <simon@pushface.org>
Date: Mon, 16 Jul 2012 08:25:12 +0100
Subject: Re: Non-GPL xml library ?
Newsgroups: comp.lang.ada

Matreshka [1] supports SAX processing.

[1] http://forge.ada-ru.org/matreshka

From: Vadim Godunko <vgodunko@gmail.com>
Date: Tue, 17 Jul 2012 00:59:40 -0700
Subject: Re: Non-GPL xml library ?
Newsgroups: comp.lang.ada

> "Can you decouple the XML part of Matreshka from the rest of the project?"

Yes, you can.

From: Maxim Reznik <reznikmm@gmail.com>
Date: Wed, 18 Jul 2012 04:45:25 -0700
Subject: Re: Non-GPL xml library ?
Newsgroups: comp.lang.ada

> "Is it widely used?"

We use it (strings API, XML read and write API, SQL API). It's nice. Now we are implementing an Ada code generator from UML using Matreshka's Ada Modelling Framework.

Sound recording API for Linux

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: 17 Jul 2012
Subject: Ada sound recording API
URL: http://repositories.jacob-sparre.dk/alsa-binding

Implemented features:

+ 16-bit mono recording (Linux)
+ 16-bit mono playback (Linux)
+ 16-bit stereo recording (Linux)

[With WAV file recording demonstration applications. —sparre]

VTKAda

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Fri, 27 Jul 2012 07:46:31 -0700
Subject: ANN: VTKAda version 5.10 free edition release 01/08/2012
Newsgroups: comp.lang.ada

I'm pleased to announce VTKAda version 5.10 free edition release 01/08/2012.

VTKAda is an Ada-2012 port to VTK (Visualization Toolkit by Kitware, Inc) and Qt4 application and UI framework by Nokia.

Package was tested with GNAT GPL 2012 (-gnat12 option) in Windows XP Sp3 32bit, Windows 7 Sp1 64bit, Fedora16 and Debian 5 x86.

As a role Ada is used in embedded systems, but with VTKAda(+QtAda) you can build any desktop applications with powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, internet browsing and many others things.

Current state of VTKAda is 42064 procedures and function distributed in 672 packages. 135 examples.

Current state of QtAda is 11925 procedures and function distributed in 324 packages.

There are many new packages and examples in this release.

VTKAda you can use without QtAda subsystem.

QtAda is an Ada port of the Qt4 framework and can be used as an independent system.

VTKAda and QtAda for Windows and Linux (Unix) free edition are available from http://users1.jabry.com/adastudio/index.html

As a role, Ada is used in embedded systems, but for desktop applications are used C/C++/C# or JAVA. With VTKAda(+QtAda) all project (included desktop part) can be written on pure Ada-2012.

Ada source code obfuscator

From: Oliver Kellogg <okellogg@users.sourceforge.net>
Date: Sun, 29 Jul 2012 07:57:30 -0700
Subject: Re: Ada source code obfuscator update
Newsgroups: comp.lang.ada

After the release of version 0.6, a number of bugs were found and fixed so I decided to release version 0.7, see http://www.okellogg.de/x.html

This version was successfully used on a system consisting of 180,000 lines of Ada code.

Changes wrt version 0.6 are:

- Extend @do_not mangle and add a note about its incompleteness.
- In sub mangled_name, add the name component onto $result unmangled if no mangling was performed on it.
- Rename sub pkg_mname to unit_mname and rename @packages to @units.
- At sub nexttoken, add optional arg $join_compound_name (default: false; this is set true on processing unit names)
- New sub skip to_first_of permits skipping to any of multiple given tokens; skipping ends on encountering the first of the given tokens.
- In main program:
  - fix iteration over @lex by replacing the "for" loop incrementing $lndx by a "while" loop employing sub nexttoken
  - fix bug in processing of task and protected declarations (the comparison against 'body' was broken)
  - detect keyword "use" so that "use type" is out of the way (otherwise the type declaration circuitry is erroneously triggered.)
- In sub wregex, change search pattern to exclude preceding ' (tic) to avoid substituting attributes.
- In sub unit_mname and main program, change unit prefix to "U".

IDE for newcomers to Ada

From: Dufr <dufriz@gmail.com>
Date: Tue, 31 Jul 2012 04:26:19 -0700
Subject: Which compiler / IDE do you recommend for a beginner?
Newsgroups: comp.lang.ada

Which compiler / IDE do you recommend for a beginner?

As I understand it, the top priority for the beginner would be choosing an environment conducive to learning. This requires of course requires reduced complexity, and possibly well explained error messages.

Which Ada compiler / environment is the best in this regard?

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Tue, 31 Jul 2012 11:36:09 -0400
Subject: Re: Which compiler / IDE do you recommend for a beginner?
Newsgroups: comp.lang.ada

I think GNAT gives very good error messages. In my opinion they are significantly above average compared to messages from other compilers I've used.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 31 Jul 2012 17:42:32 +0200
Subject: Re: Which compiler / IDE do you recommend for a beginner?
Newsgroups: comp.lang.ada

[…]

Give GPS a second try. It is very good and more productive to use than AdaGide (which is very nice too and was the first choice before GPS matured).

One thing about GPS. Create and edit your project files manually you will have no problems whatsoever.

From: francois_fabien@hotmail.com
Date: Tue, 31 Jul 2012 11:02:36 -0700
Subject: Re: Which compiler / IDE do you recommend for a beginner?
Newsgroups: comp.lang.ada

GNAT/GPS (Adacore has a GPL version) is a nice and matured IDE to start with. At the start, you must set up project, but the wizard is very helpful.

To get acquainted with the language you have plenty (500+) of code samples at RosettaCode [1].

When you will have bigger projects with many libraries, the remark of Dmitry is true: you must handle project files outside GPS.

[1] http://rosettacode.org/wiki/Category:Ada

From: Britt <britt.snodgrass@gmail.com>
Date: Tue, 31 Jul 2012 18:55:38 -0700
Subject: Re: Which compiler / IDE do you recommend for a beginner?
Newsgroups: comp.lang.ada

[…]

gpr files are easy to understand and very powerful (just read the users guide). A brief (2 to 10 line) project file may be all you initially need. It would be a Good Thing if other Ada vendors would adopt a gpr-like project structure.

From: Nicholas Paul Collin Gloucester <Colin_Paul_Gloster@acm.org>
Date: Thu, 2 Aug 2012 11:12:13 +0000
Subject: Re: Which compiler / IDE do you recommend for a beginner?
Newsgroups: comp.lang.ada

PowerAda from OC Systems.

LAPACK and BLAS binding

From: Nasser M. Abbasi <nma@12000.org>
Date: Tue, 31 Jul 2012 01:37:05 -0500
Subject: fyi, small update to Ada LAPACK and BLAS binding
Newsgroups: comp.lang.ada

I've added more documentation and made a little cleanup of the current Ada LAPACK and BLAS bindings.

As per earlier thread, this snap shot of the LAPACK binding now uses one package to interface to LAPACK so it is easier to use.

The location is still the same as before, and with more documentation now how to use the bindings.

http://12000.org/my_notes/ada/index.htm

I have a zip file the LAPACK and BLAS updates I made there with links to the original versions

AVR-Ada 1.2 release candidate

From: Rolf Ebert <rolf.ebert.gcc@gmx.de>
Date: Wed, 01 Aug 2012 21:41:10 +0200
Subject: Tentative release of v1.2.0
Mailing-list: AVR-Ada <avr-ada-devel@lists.sourceforge.net>

I am currently uploading the source and windows binary release V1.2.0.

As I will leave tonight for business and holiday trips, I won't be able to correct any issues. Nevertheless I encourage you to try the new version and report any problems.

Although the new wiki is quite a mess right now, I also invite you to document your experience and install issues in the wiki.

GtkAda contributions

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 11 Aug 2012 08:31:33 +0200
Subject: ANN: GtkAda contributions v2.14
Newsgroups: comp.lang.ada

The library is a contribution to GtkAda, an Ada bindings to GTK+ toolkit. It deals with the following issues: tasking support; custom models for tree view widget; custom cell renderers for tree view widget; multi-columned derived model; an extension derived model (to add columns to an existing model); an abstract caching model for directory-like data; tree view and list view widgets for navigational browsing of abstract caching models; file system navigation widgets with wildcard filtering; resource styles; capturing the resources of a widget; embeddable images; some missing sub-programs and bugfixes; a measurement unit selection widget and dialogs; an improved hue-luminance-saturation color model; simplified image buttons and buttons customizable by style properties; controlled Ada types for GTK+ strong and weak references; and a simplified means to create lists of strings.

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

[see also "GtkAda contributions ", in AUJ 33-2 (June 2012), p.77 —sparre]

Industrial control widget library

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 11 Aug 2012 08:39:57 +0200
Subject: ANN: Ada industrial control widget library v1.4
Newsgroups: comp.lang.ada

AICWL is an Ada library that is intended for designing high-quality industrial control widgets for Ada applications. The widgets are composed of transparent layers drawn by cairo. The widgets are fully scalable graphics. A time controlled refresh policy is supported for real-time and heavy-duty applications. The library supports caching graphical operations and stream I/O for serialization and deserialization. Ready-to-use gauge and meter widgets are provided as samples as well as an editor widget for WYSIWYG design of complex dashboards. The software is based on GtkAda and cairoada, the Ada bindings to GTK+ and cairo.

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

Simple components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 11 Aug 2012 08:19:51 +0200
Subject: ANN: Simple components for Ada v3.19
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, directed graphs, sets, maps, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support. Tables management and strings editing are described in separate documents see Tables and Strings edit. The library is kept conform to both Ada 95 and Ada 2005 language standards.

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

Deepend

From: Brad Moore <brad.moore@shaw.ca>
Date: Mon, 27 Aug 2012 23:17:24 -0600
Subject: ANN: Deepend 3.2 Storage Pools
Newsgroups: comp.lang.ada

I am pleased to announce the availability of Deepend version 3.2.

Deepend is a suite of dynamic storage pools with subpool capabilities for Ada 95, Ada 2005, and Ada 2012.

Bounded and unbounded storage pools types are provided. Storage pools with subpool capabilities allow all objects in a subpool to be reclaimed all at once, instead of requiring each object to be individually reclaimed one at a time. Deepend storage pools are more efficient and safer than other schemes and can eliminate the need for Unchecked_Deallocations. A Dynamic Pool may have any number of subpools.

Deepend can be downloaded from;

https://sourceforge.net/projects/deepend/files/

[Deepend 3.1 was announced 6 July 2012. See also “Deepend 2.6” in AUJ 32-4 (December 2011), p.216 —sparre]

AdaControl

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Wed, 29 Aug 2012 16:14:42 +0200
Subject: AdaControl 1.14 released
Newsgroups: comp.lang.ada

Adalog is pleased to announce the release of AdaControl 1.14. As usual, this release features new controls (meter says 421!), new features, bug fixes…

More importantly, it compiles with GNAT GPL 2012! A change in the ASIS interface prevented the previous version from compiling with the latest versions of GNAT. AdaControl is now provided in two flavors, one for the "old" GNAT, and one for the recent one. Features are the same, except that some controls related to Ada 2005/2012 are not available with the "old" version.

As usual, AdaControl can be downloaded from:

http://www.adalog.fr/adacontrol2.htm

Ada-related Products

GNAT GPL 2012 and SPARK GPL 2012

From: Jamie Ayre <ayre@adacore.com>
Date: Tue, 26 Jun 2012 11:56:09 +0200
Subject: [AdaCore] Announcing the availability of GNAT and SPARK GPL 2012
Mailing-list: libre-news@lists.adacore.com

Dear GNAT and SPARK GPL user,

We are pleased to announce the release of GNAT GPL 2012, the integrated Ada, C, and C++ toolset for academic users and FLOSS developers. This new edition provides many new features and enhancements in all areas of the technology. The most notable ones are:

- Full support for Ada 2012;
- Tool enhancements:
- GPS 5.1 (improved support for C/C++, centralized handling of VCS menus, improved automatic code fixes, more intuitive handling of the MDI)
- GtkAda 2.24 (bindings upgraded to Gtk 2.24, support for printing, support of the Glade-3 GUI builder and the Gtk_Builder approach)
- GNATbench 2.6 (ergonomic improvements, additional tooltip functionality)
- A dimensionality checking system for physical units
- Improvements to GNATpp and GNATmetric
- Support for unloading Ada plug-ins
- Improved Ada/C++ integration
- New warnings and better error messages
- Support for encapsulated shared libraries with no external dependencies
- New unit GNAT.Expect.TTY for advanced terminal interaction

We are also pleased to announce the release of SPARK GPL 2012, the integrated static analysis and verification toolset for academic users and FLOSS developers.

This new edition provides many new features and enhancements to both the SPARK language and toolset. The most notable ones are:

- Support for generic subprograms
- Annotated and refined proof functions
- Fully-modelled function calls in proof contexts
- Assume statements
- Full-range subtypes for all types
- Automatic data flow analysis mode
- Improved Examiner options for platform-independence and easier makefile integration
- Improved use of types & subtypes in FDL
- Improved Simplifier rules and tactics
- Extension of SPARKBridge to accommodate other SMT solvers: CVC3, Yices and Z3

GNAT GPL 2012 and SPARK GPL 2012 can be downloaded from the "Download" section on https://libre.adacore.com.

GNATprove available for download

From: Jamie Ayre <ayre@adacore.com>
Date: Mon, 23 Jul 2012 10:15:45 +0200
Subject: [AdaCore] Availability of GNATprove
Mailing-list: libre-news@lists.adacore.com

Following the recent release of GNAT GPL (see http://libre.adacore.com/) we are happy to announce the first GPL release of the GNATprove. This tool is used for formal verification of Ada programs and is being developed as part of the Hi-Lite project. We provide binary distributions for x86 Linux, x86 windows and x86-64 bit Linux. More details can be found on the following page:

http://www.open-do.org/projects/hi-lite/gnatprove/

For questions, remarks, or issues please contact us on

<hi-lite-discuss@lists.forge.open-do.org>

Atego acquires IBM Rational Ada Developer products

From: Martin Dowie <martin@thedowies.com>
Date: Wed, 25 Jul 2012 00:17:14 -0700
Subject: Atego acquires IBM Rational Ada Developer products
Newsgroups: comp.lang.ada

http://www-01.ibm.com/support/docview.wss?uid=swg21605950&myns=swgrat&mynp=OCSSMMQY&mynp=OCSSSGSH&mync=E

Further rationalization in the Ada market… (pun intended :)

From: Britt <britt.snodgrass@gmail.com>
Date: Thu, 16 Aug 2012 18:49:44 -0700
Subject: Re: IBM (Rational) Ada now owned by Atego!
Newsgroups: comp.lang.ada

[http://www-01.ibm.com/software/rational/info/apex-ada —sparre]

I think it is good news since Apex was a very good (though costly) development environment. I used it from 1996-2002. However IBM hadn't changed it much in recent years. I hope Atego can revitalize Apex and make it competitive again. While I'm currently a great fan (and supported customer) of AdaCore and GNAT Pro, I know it is not good for the Ada language and the Ada user community to have only one healthy compiler vendor.

AdaCore has taken over the commercial Ada market because they (in my opinion) do everything right with their support subscription based business model, their proactive development of supporting

technologies, their early implementation of new Ada standards, and by strongly promoting the Ada language itself rather that just selling tools. If not for what AdaCore has done over the last 17 years or so, there would be much less use of Ada today. So while I wish Atego success with both Apex and ObjectAda, I don't think they will regain much market-share unless they choose to become more like AdaCore. If they do then I think there will be enough business for all.

From: Martin Dowie <martin@thedowies.com>
Date: Fri, 17 Aug 2012 00:38:09 -0700
Subject: Re: IBM (Rational) Ada now owned by Atego!
Newsgroups: comp.lang.ada

Wouldn't it be bizarre (but rather brilliant) if Atego opened up ObjectAda and/or Apex to the single-user/small company market? How about a $100 / £100 / Euro100 per year subscription model?.

GNATprove distinguished at VerifyThis competition

From: Yannick Moy
Date: September 3, 2012
Subject: GNATprove Distinguished at VerifyThis Competition
URL: http://www.open-do.org/2012/09/03/gnatprove-distinguished-at-verifythis-competition/

I participated last week in the VerifyThis Verification Competition, which took place on Thursday afternoon during the Formal Methods 2012 conference in Paris. The goal was to apply verification tools to three small challenge programs, to compare approaches and learn from each other's tools.

I used Ada 2012 as a programming and specification language (using preconditions and postconditions to specify contracts for subprograms) and our prototype GNATprove, a proof tool developed in project Hi-Lite, to formal verify that the code implements its contract and does not raise run-time errors (integer overflows, array index out of bounds, etc.) I completed challenge 1 and I did a part of challenge 2, but I had not enough time to complete it or start on challenge 3.

The competition was followed on Friday by a very interesting explanation session where each team showed how it addressed the problems with its tools. It was particularly interesting to see different solutions from teams using the same language (for example, the two teams using Why3 had quite different solutions for challenge 2), as well as the interaction between the user and the proof tool in KIV, KeY, Why3, etc. I think the problems and their solutions will be added soon to the VerifyThis repository, but if you cannot wait, you can also ask the organizers for a tarball of the submissions.

To come to the title of this post, the organizers awarded a distinction to GNATprove for its integration of proving and run-time assertion checking, of which I'm very proud. As I explained them, this integration was essential in helping me during the competition:

* For the first problem, I was stuck with a postcondition that I could not prove, and I did not manage to figure out why. So I decided to write a small test to make sure at least that the code and the contract were not contradictory. I executed it, and it raised an exception saying the postcondition was wrong! (because Ada 2012 contracts are executable, the compiler can transform them into run-time assertions, including quantifiers that are transformed in loops) It was then easy to pinpoint the root cause of the problem, the use of "<" instead of "<=" in the test of the main loop.

* For the second problem, I decided to implement the iterative version of the algorithm, which is more complex to specify and verify than the recursive one, but also more representative of critical embedded software. The algorithm is divided in two passes, each one performing two nested loops on the input array, with loop invariants to write for the proof to go through. Being able to execute these loop invariants as regular assertions made me quite confident that I had not written wrong assertions, before I even start proving something.

Hope to see even more participants at the next software verification competition, either VSTTE's one or VerifyThis!

Ada and Mac OS X

GtkAda on Mac OS X

From: Emmanuel Briot <briot@adacore.com>
Date: Thu, 26 Jul 2012 09:22:59 +0200
Subject: Re: Patch: Remove a remnant of the Glade support.
Mailing-list: gtkada@lists.adacore.com

We do check that GtkAda works on OsX (in fact most of the GtkAda developers are on that platform), but this isn't an officially supported platform and building a release requires a lot of resources. We are indeed very interested in user feedback for various aspects of GtkAda.

In fact, the API in gtk+3 has change significantly, and of course so has the one in GtkAda. Porting is not difficult, but is not a simple matter of recompiling either. So this is a good opportunity for us to improve areas of GtkAda that haven't changed in the last 12 years. In particular, we are rethinking the handling of signals for instance.

OS X 10.8 Mountain Lion and GNAT GPL 2012

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Mon, 30 Jul 2012 20:51:40 +0100
Subject: OS X 10.8 Mountain Lion and GNAT GPL 2012
Newsgroups: comp.lang.ada

Mountain Lion needs Xcode 4.4, which does not install by default the command line tools /usr/bin{as, ld, make} needed by GNAT.

It is necessary to invoke:

Xcode > Preferences… > Downloads > Command Line Tools: Install

It's about a 115MB download.

Once completed, GNAT GPL 2012 is restored to sanity.

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Mon, 30 Jul 2012 23:40:41 +0100
Subject: Re: OS X 10.8 Mountain Lion and GNAT GPL 2012
Newsgroups: comp.lang.ada

I suspect the GCC tools are now an optional download because Apple have adopted LLVM in place of GCC as the standard compilation toolset. I can't see them ever making it impossible to use GCC.

GtkAda and more for Snow Leopard

From: Pascal <p.p14@orange.fr>
Date: Sun, 2 Sep 2012 18:37:45 +0200
Subject: [ANN] XAdaLib binaries for SL including GtkAda and more.
Newsgroups: gmane.comp.lang.ada.macosx

This is XAdaLib 2012 built on Mac OS X Snow Leopard for X11 including:

- GTK Ada 2.24.2 with GTK+ 2.24.5 complete,
- Glade 3.8.2,
- GNATColl 2012,
- Florist 2012,
- AICWL 1.3,

Then see documentation and examples in share directory and enjoy.

See the instructions which have produced the libraries on Blady web site:

http://blady.pagesperso-orange.fr/creations.html#gtkada

XAdaLib binaries have been post on Source Forge:

http://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2012-snow-leopard/

References to Publications

New SPARK book

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Date: Fri, 29 Jun 2012 09:51:37 -0700
Subject: New SPARK book available for pre-order
Newsgroups: comp.lang.ada

The new SPARK book "SPARK: The Proven Approach to High Integrity Software" seems to be available for pre-order at Amazon UK. Publication date is stated to be July 2012.

http://www.amazon.co.uk/Spark-Proven-Approach-Integrity-Software/dp/0957290500

A New Language for Safe and Secure Software

From: Benjamin M. Brosgol
Date: July 2012
Subject: Ada 2012: A New Language for Safe and Secure Software
URL: http://cotsjournalonline.com/articles/view/102810

Building on a tradition of success in mil/areo systems, a new version of the Ada language has emerged. It features "contract-based programming" that blends well with the requirements-based world of military programs.

[…]

Ada Inside

The ideal programming language?

From: Colin Walls
Date: Aug 1, 2011
Subject: The ideal programming language?
URL: http://go.mentor.com/w6n9

I recently wrote about programming languages and discussed which ones are common for embedded applications. Among the responses to that posting was a message from Robert Dewar from AdaCore. He makes the following points:

"It's interesting to note that you omitted one language mentioned in the VDC report, namely Ada. Ada is interesting because, as you noted, all the other languages do not share Ada's important characteristic that it was designed for embedded use. For example, the recognition that multi-threading is fundamental in most embedded programming, and therefore comprehensive features for handling this are an important part of the Ada language design. Other languages have no support at all for threading in the language itself (C, C++), or very rudimentary support (Java) that falls far short of what is needed (a fact recognized by the attempt to extend Java for real time use).

The usage of Ada mentioned in the VDC report is listed as 3%, but that number may be deceptive in that it misses the fact that Ada's predominant use is in large-scale safety- and security-critical systems. Significant parts of the avionics of many new planes are written in Ada, and also a number of other critical systems, such as air traffic control systems. I suspect that if you counted lines of code, and weighted the results by the significance of the applications involved, Ada would come quite a bit higher on the list."

Robert is, of course, completely correct. Although not unique in its incorporation of multi-threading in the language, Ada is probably the only widely used language that was really designed for embedded applications. I think that my own lack of experience with Ada meant that I did not give it due consideration. I wonder if Ada usage is declining, stable or growing?

SmartSide Adopts Ada and GNAT Pro for Smart Devices Platform

From: AdaCore Press Center
Date: June 20, 2012
Subject: SmartSide Adopts Ada and GNAT Pro for Smart Devices Platform
URL: http://www.adacore.com/press/smartside-adopts-ada-gnat-pro/

PARIS, NEW YORK, June 20, 2012 - SG PARIS 2012 Conference - AdaCore today announced that SmartSide, a Paris-based company providing Smart Metering and Smart Grid management solutions, has adopted the Ada programming language and AdaCore's GNAT Pro development environment for the implementation of their Smart Devices platform. SmartSide offers multi-energy meter data management systems. Distribution Network Operators use SmartSide technology to optimize their Smart Grid networks through the secure, reliable, highly-interoperable and business-oriented Smart Energy Core platform.

"In our relentless search for quality and performance, Ada has come up as the most efficient technology for writing reliable, secure and scalable code that is also easily maintainable," said David Dhéux, SmartSide CTO. "GNAT Pro is our framework of choice for writing Ada code. Its all-in-one development environment allows us to handle activities ranging from initial development to static analysis and testing. It reduces our time-to-market and gives us the edge we need in today's competitive environment. We also selected AdaCore for the high quality and responsiveness of its support team. Finally, we chose to work with AdaCore because their product roadmap is particularly well-suited to ours, with tools and language evolution focused on reliability."

The system consists of two primary components:

* The generic core, which acts as the intelligence of the system and performs the major work (data collection, processing, analysis). This part of the system is sufficiently generic and configurable to process all kinds of energy and environmental data.

* The user interfaces, which are separate from the core, are very flexible and easy to adapt. They are easily configurable to reflect specific business needs.

The infrastructure of the Smart Devices platform is designed to be scalable, reliable and fault resistant. GNAT Pro and the Ada programming language were chosen because of their long and successful track record in the aerospace and defense industries, where high levels of reliability are critical. SmartSide thoroughly evaluated several other languages, but none matched the reliabilty and data-handling qualities of Ada.

"SmartSide's diligence in searching for an appropriate programming language led them to Ada, which is especially pleasing," said Jamie Ayre, Marketing Director of AdaCore Europe. "Ada has made a name for itself as a language for programming reliable, safe and secure systems. With the advent of Ada 2012 and the new features it introduces, it has become the benchmark for the development of these systems."

Telecom ParisTech third in robotic cup

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 22 Jun 2012 19:03:54 +0200
Subject: Telecom ParisTech third in robotic cup
Newsgroups: comp.lang.ada

against 140, and you guess it, their robot is in Ada-Ravenscar.

See f.e. http://libre.adacore.com/academia/projects-single/robotics-cup.

If you want to see the pictures, you'll have to go through sites in French… (google "telecom robotique")

TeXCAD

From: Gautier gautier_niouzes@hotmail.com
Date: Fri, 6 Jul 2012 06:55:54 -0700 (PDT)
Subject: Ann: TeXCAD 4.3
Newsgroups: comp.lang.ada

After a long time of inactivity, here is the 4.3 version of TeXCAD, a 100% Ada software.

http://texcad.sf.net

TeXCAD is a picture editor for pure LaTeX or, at will, some extensions.

No support yet for fancy packages like PSTricks or TikZ, but perhaps, one day…

On the other hand, TeXCAD allows to make very portable vectorial pictures, and even to rework pictures made with some no more supported extensions like emlines and save them without those extensions. The TeXCAD underlying library is OS-independent and can be used to make batch converters to various mix of LaTeX graphics packages, or to apply various graphics user interface layers.

AWS inside

From: Pascal Obry <pascal@obry.net>
Date: Sat, 07 Jul 2012 09:38:07 +0200
Subject: Re: so talk to me about aws: anyone got a site? any e commerce? fun?
Newsgroups: comp.lang.ada

A photo critic Web site (in French):

http://v2p.fr.eu.org

This is based on AWS using most of the fun stuff in it, it was a test bed for the Web_Block support.

It is OpenSource: https://github.com/TurboGit/vision2pixels

From: Manuel Gomez <mgrojo@gmail.com>
Date: Mon, 09 Jul 2012 20:59:22 +0200
Subject: Re: so talk to me about aws: anyone got a site? any e commerce? fun?
Newsgroups: comp.lang.ada

Since this is a recurrent question I have compiled a list of sample websites implemented in AWS and added it to AdaCommons wiki. Everyone is invited to add any other site to the list.

http://commons.ada.cx/Ada_Web_Server

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Thu, 12 Jul 2012 14:18:17 -0700
Subject: Re: so talk to me about aws: anyone got a site? any e commerce? fun?
Newsgroups: comp.lang.ada

Perhaps not fun, but I find them cool

<http://www.youtube.com/watch?v=3DiyVDMp2bL9c>

I communicate with them via AWS, http POST. No I don't control each one of them independently, but provide them with assignments, like 'take bin 1, 3, 4, and 5 and put them in port 2. tell me when you deliver each one of them.'

Our installation has conveyors at the port, giving the pickstation a 10 bin buffer.

This is within a Warehouse Control System.

We also use AWS as a web server within our Warehouse management system, making the operator at infeed take a photo of bad/broken pallets at goods reception with a handheld device, uploading via AWS, for further treatment in WMS/ERP -i.e. tell the transporter to be more careful.

At some site we, use AWS as a message broker, providing web-service interface in one end, and inhouse format in the other end.

Great product.

We use it on Windows and AIX.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 13 Jul 2012 10:03:59 -0700
Subject: Re: so talk to me about aws: anyone got a site? any e commerce? fun?
Newsgroups: comp.lang.ada

"Is the whole system (or some high proportion) Ada?" The autostore system itself is probably c :-((Would be nice to redo it in Ada though)

The Warehouse Control System (WCS), that tells them to do, on a higher level is all Ada. That system talks to conveyors (usually PLCs) stacker cranes, labelizers, AGVs, LGVs (auto trucks), selects locations to store goods within a warehouse and of course talks to WMS/ERP systems. (where we use AWS sometimes) In this case we have

ERP|WMS|WCS|Conveyer + Autostore where the WCS is all Ada. Sattmate WCS in Ada

<http://www.consafelogistics.com/Our%20offer/Warehouse%20Management/SattMate%20WCS> there is also a combined WCS/WMS that is all Ada (Except the gui, in both cases) SattStore, WMS/WCS in Ada

<http://www.consafelogistics.com/Our%20offer/Warehouse%20Management/SattStoreWMS>

Sledgehammer Indexing Tools

From: Erich <john@peppermind.com>
Date: Mon, 6 Aug 2012 11:36:03 -0700
Subject: Sledgehammer Indexing Tools
Newsgroups: comp.lang.ada

I've just released some document indexing tools for GNU/Linux on launchpad which are written in Ada:

https://launchpad.net/sledgehammer

They come with a makefile and a GPS project file. They heavily depend on external converters, so please check the dependencies in the Readme file before trying the indexer out.

Comments and suggestions for improvement are welcome. Please bear in mind that I'm a hobbyist and this is only the second Ada program I've ever written, though, so please don't be too harsh with your criticisms. :-)

Perhaps someone finds part if this useful.

Adagio

From: Shark8 <onewingedshark@gmail.com>
Date: Sun, 12 Aug 2012 13:22:04 -0700
Subject: Ada networking (Adagio)
Newsgroups: comp.lang.ada

Hey, I was hoping to get a bit more familiar with using Ada for networking programs; I found referenced to Adagio, which is supposed to be (or have been, it's inactive now) a Gnutella2 program.

The project is inactive and none of the files appear to have been archived/downloadable. The last known source was contained in a zipfile adagio-src.2.1.e.zip, which I cannot find for download either.

I was wondering if anyone on this thread might have a copy laying around on their HD.

From: Mosteo <alejandro@mosteo.com>
Date: Mon, 20 Aug 2012 10:59 +0200
Subject: Re: Ada networking (Adagio)
Newsgroups: comp.lang.ada

I'm not sure how good a resource it is for learning from; I did learn a lot about blocking and non-blocking sockets when programming that (I experimented with both approaches -- IIRC a stack size related thread limit in Windows prompted me to abandon blocking IO in the end), but the code more or less grew up with little foreplanning.

From: Mosteo <alejandro@mosteo.com>
Date: Wed, 29 Aug 2012 13:08:11 +0200
Subject: Re: Ada networking (Adagio)
Newsgroups: comp.lang.ada

I've rescued my last version. The story seems missing though. I've made a push to github at:

https://github.com/mosteo/adagio

Project file is adagio.gpr. There are others for related projects that never got anywhere, so I'm leaving these for later.

From: Mosteo <alejandro@mosteo.com>
Date: Thu, 30 Aug 2012 11:18:15 +0200
Subject: Re: Ada networking (Adagio)
Newsgroups: comp.lang.ada

> Not surprising. I'm not sure if sockets are the best way to handle networking in-general -- though they certainly are the most popular.

I've used Yarp (mandated in a project) and came to hate it. But I'm pretty sure that there are better middlewares out there. From the point of view of ready for use with Ada I'm curious about YAMI4 and zeromq.

DrDobbs Longing For Code Correctness

From: Andrew Binstock <alb@drdobbs.com>
Date: August 26, 2012
Subject: Longing For Code Correctness

URL: http://www.drdobbs.com/architecture-and-design/longing-for-code-correctness/240005803

Longing For Code Correctness

By Andrew Binstock, August 26, 2012

The longer I write code, the more I yearn for code correctness. Despite the work this extra step presents, commercial ventures, especially Wall Street, would do well to embrace it.

When I was young and first got into programming seriously, I reveled in working at the lowest levels of code. I loved (loved!) assembly language and could think of no greater happiness than doing clever things with registers and writing tight, fast code that squeezed into the minimum amount of RAM and delivered great performance. The second alternative is to use languages that strongly support correctness.

There are not many. Ada goes farther than Java in this regard. And Eiffel perhaps farther yet. But probably the one that does the most is SPARK, an Ada subset. Developed in the U.K., SPARK uses a system of annotations that are embedded in comments at the start of each method. The code can then be verified by assorted tools that check it against the annotations. Free and commercial versions of SPARK tools exist today.

[Andrew Binstock is Editor-in-Chief at DrDobbs —sparre].

Ada in Context

Ada to C translator for small microcontrollers

From: Tomi Saarnio <kalvin.news@gmail.com>
Date: Mon, 26 Mar 2012 05:48:47 -0700
Subject: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

As there exists at least one free decent C cross-compiler for smaller 8-bit and 16-bit architectures (Pic, Avr, 8051 etc.), but none Ada compiler that I know of, I was wondering whether there is a demand for an Ada to C translator, that would implement some sort of Ada subset and output corresponding ANSI C source code. I am not a compiler or Ada expert, so I cannot really estimate the effort how hard this would be.

As far as I can see the problem, the first thing is to identify a suitable subset(s) to be implemented, and then to come up with the corresponding C source idioms.

From: Rego, P. <pvrego@gmail.com>
Date: Mon, 26 Mar 2012 06:44:51 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

For AVR, now you know…

http://libre.adacore.com/libre/download2?config=avr-elf-windows&version=2011

Actually you also have AVR-Ada. For RTOSes, RTEMS and MarteOS have support for Ada.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Mon, 26 Mar 2012 17:14:00 +0300
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

SofCheck (www.sofcheck.com) provides such an Ada (95) compiler, a version of their AdaMagic front-end.

“I am not a compiler or Ada expert, so I cannot really estimate the effort how hard this would be.”

Quite large, I think, unless you take a very small subset. But one would really like the subset to provide most of the compile-time advantages of Ada: packages, user-defined types, type attributes, generics.

I think the realistic approach for an open-source implementation would be to take the existing GNAT front-end and try to generate C code from the intermediate representation, either the gcc IR or the llvm IR.

In the GNAT-based approach that I suggest above, the bottleneck is in the IR-to-C translator. I don't know much about the gcc or llvm IRs, but I would expect the subset to be defined by the limitations of whatever IR-to-C translator is created, and it may not be easy to define the corresponding subset on the Ada level. In any case, it seems simpler to start building an IR-to-C translator and accept whatever limitations it turns out to have.

There are some "Ada to C/C++" translators that work on the pure source-to-source "idiom" basis, but I believe they are intended to help porting Ada projects from Ada to C or C++, and probably require manual assistance to finish the translation. The SofCheck tool is a fully automatic, real compiler that generates complete and finished C source.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 26 Mar 2012 08:48:55 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

That's a good thought. LLVM already comes with a C backend (i.e. a code generator that emits C rather than assembly). I wonder how easy it would be to configure a toolchain based on the GNAT Ada front-end and this C back-end, bound together by LLVM's DragonEgg GCC plug-in.

From: Simon Wright <simon@pushface.org>
Date: Mon, 26 Mar 2012 17:20:27 +0100
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

Hmm, from [1], "The C backend has numerous problems and is not being actively maintained. Depending on it for anything serious is not advised."

[1] http://llvm.org/docs/ReleaseNotes.html#knownproblems

From: kalvin.news@gmail.com
Date: Tue, 27 Mar 2012 02:46:00 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

[...]

There is also GPL Ada to C/C++ translator[1], but I have not looked at this yet. It seems that this tool is mainly aimed to aid the process of translating existing Ada source code to C/C++. However, the GPL'd source code is available.

[1] http://adatoccpptranslator.free.fr/

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Wed, 28 Mar 2012 19:36:37 +0300
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

“Full language translators are mostly used to move a project away from one language to another more accessible language. So, why use Ada in the first place.”

Because it is so much better than C, of course :-)

I know of more than one project that used Ada as a high-level design and specification language, and then implemented the design in some lower-level language (in one case, an assembly language). In some cases, the Ada code was complete and runnable and could be used as a test oracle for the lower-level implementation.

“How large a software project needs to be in order to benefit from implementing it in Ada?”

IMO, basically there is no lower limit, with the possible exception of the null program.

“Why bother with Ada as we already have (free) C compiler available.”

We also have a free Ada compiler available, for many platforms -- but not for many small processors, I grant.

But many projects developing in C for microcontrollers use non-free, commercial compilers, IDEs, and other expensive tools (in-circuit debuggers, etc.) The question IMO is not whether there is a benefit from using the Ada language; the question is if the benefit can be convincingly quantified in money and schedule terms.

“If the project is a small one, and you don't have to share the code with others, C is just fine.”

No! C is a like a Model-T Ford that for some strange reason is allowed on modern roads. You can survive and not be too uncomfortable on short trips, but you should really consider changing to a better car.

IMO the only reasons that would make me use C instead of Ada are:

- No Ada compiler available within my budget

- Need to use large C libraries/APIs for which no Ada binding exists.

From: KK6GM <mjsilva@scriptoriumdesigns.com>
Date: Wed, 28 Mar 2012 10:56:34 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

[…]

The free vs not-free issue should not be discounted. People who would be open to trying Ada on small platforms need to be able to get it into their hands easily. Free now to help make the sale, pay later for support. It would be fabulous, IMO, if the SofCheck Ada->C product would be made available in some free form. I know I would start using it immediately, and start trying to convert my organization to Ada. There is so much sub $5-$10 hardware now that could run Ada code very effectively, but continues to be programmed in C/C++ in part because of the inertia of the free or very low cost compilers available.

[…]

"The 1980s will probably be remembered as the decade in which programmers took a gigantic step backwards by switching from secure Pascal-like languages to insecure C-like languages. I have no rational explanation for this trend." -Per Brinch Hansen.

From: KK6GM <mjsilva@scriptoriumdesigns.com>
Date: Wed, 28 Mar 2012 07:29:41 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

[…]

Even the smallest microcontroller projects could greatly benefit from Ada's real-time and concurrency features.

From: kalvin.news@gmail.com
Date: Sat, 31 Mar 2012 08:46:08 -0700 (PDT)
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

Yes, the translator should be implemented so, that it would be compatible with the target C ie. no support for 64-bit integers etc. if the target C compiler doesn't support that.

Nested procedures are also a bit tricky to implement in C, as the local stack frame needs to made visible to the inner procedure. This is doable with the struct of pointers passed as parameters for the "inner procedure", I guess. It is up to target C compiler optimization how much penalty there will be.

From: kalvin.news@gmail.com
Date: Sun, 1 Apr 2012 04:23:37 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

P2C (Pascal-to-C) might also be a viable starting point for the translator.

However, in order to take advantage of mature GNAT compiler front-end, the GCC looks tempting path. GCC seems to be quite complicated beast, but maybe it can be tamed for the purpose.

From: BrianG <me@null.email>
Date: Mon, 02 Apr 2012 22:08:40 -0400
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

If the target has a GCC port, it is not that difficult to create a cross-compiler from GNAT, using No_Run_Time (i.e. no tasking, etc; you have to build your own binding to the target libraries or registers). I did this once for a uC board I have there was an in-work Ada project at the time, but I only used their instructions for building the cross-compiler). I have no compiler or GCC experience, and was able to get it working - not sure if I could, or want to, do it again.

I wonder if it's possible to create a C-target back end? :-)

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Tue, 03 Apr 2012 11:29:39 +0200
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

“I wonder if it's possible to create a C-target back end? :-)”

Won't it be a lot easier, and more flexible for industry, if there is some commonly accepted intermediate language for µControllers that supports C well, but also supports other languages?

The intermediate language then removes the need (and cost!) for implementers to implement the complicated C stuff correctly.

OTOH, the makers of C compilers will not be happy, I guess, if there is a solution that makes switching languages easy. Oh well, I guess the makers of Ada compilers will feel the same.

From: kalvin.news@gmail.com
Date: Mon, 21 May 2012 03:35:51 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

I found this paper "No Assembly Required: Compiling Standard ML to C"[1] which presents some ideas and experience in using the C as compiler target language.

The Standard ML differs quite a lot from C, and "The generated code achieves an execution speed that is about a factor of two slower than a native code compiler". However, as Ada is closer to C, the resulting overhead is supposed be less. Also, implementing only a carefully selected subset of Ada language might help in creating better translation to target C language.

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.154&rep=rep1&type=pdf

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Mon, 21 May 2012 14:27:07 +0200
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

Eiffel also generates C; there is a GPL edition of the compiler. Don't know whether it is suitable for µControllers, though.

From: Marco
Date: Sat, 2 Jun 2012 08:27:06 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

SmartEiffel

http://en.wikipedia.org/wiki/SmartEiffel

http://smarteiffel.loria.fr/

Could be used for ideas on creating a subset Ada to C "compiler".

From: kalvin.news@gmail.com
Date: Tue, 5 Jun 2012 02:18:11 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

As Niklas Holsti suggested sometime above, the translation from GNAT IR to C might be the easiest way. I thought about this, and as there exists a GNAT Ada Pretty Printer, and if I have understood it correctly, it uses GNAT IR for recreating the source code. This might be the easiest way to create the translator, as rewriting the source code formatting rules to produce C source instead.

From: Marco <prenom_nomus@yahoo.com>
Date: Sun, 10 Jun 2012 08:41:38 -0700
Subject: Re: Ada to C translator for small microcontrollers
Newsgroups: comp.lang.ada

Create a small Ada program with some short procedures and functions (skip tasks and IO) and try to "hack up" the GNAT Ada Pretty Printer to output equivalent C to see if this is a viable option forward. At

the very least you will learn more about GNAT.

Ada.Storage_IO

From: Micronian Coder <micronian2@gmail.com>
Date: Sat, 9 Jun 2012 01:18:05 -0700
Subject: Ada.Storage_IO. Anyone use it?
Newsgroups: comp.lang.ada

I was looking through the Ada RM and came across the generic package Ada.Storage_IO. I'm not sure what the benefits are to using it. It states that it could automatically flatten out the data representation if necessary. For example, say a discriminant record has an array field that is sized based on the discriminant value:

```
type Buffer(Length : Positive) is
   record
      Bytes: Byte_Array(1 .. Length);
   end record;
```

The implementation chosen by the compiler could have Bytes dynamically allocated rather than embedded (I believe Randy said the Janus/Ada compiler would always have the arrays of this type of record dynamically allocated). Using Ada.Storage_IO would store all the data into a buffer as if it was all together and when read back it could recreate the multi-part representation.

Did anyone find this package useful in practice rather than define their own IO routines (e.g. define stream operations to read and write their data)? I'm really curious.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 9 Jun 2012 10:36:39 +0200
Subject: Re: Ada.Storage_IO. Anyone use it?
Newsgroups: comp.lang.ada

You cannot instantiate Ada.Storage_IO with this type, because Buffer is indefinite. But since Ada.Storage_IO look totally useless (see A.9(11), too!) anyway…

[…]

1. For specifically storage I/O I am using memory pools. Instead of Write I do "new", instead of Read I use an access type. This is cleaner, more efficient, and no generics involved.

= Pool, backed by a container, e.g. a segmented memory stack

2. For data exchange I am using streams. I always redefine stream operations because built-in ones are unusable for data exchange, which dictates certain representation. Again, no generics, no any limitations of elements, reusable with different stream backends.

= Stream, backed by a container.

It is worth to mention that there are three major distinct cases, which get permanently confused in the context of I/O:

A. Marshaling objects (true I/O, persistency, serialization)

B. Formatted I/O and rendering (dealing with human readable representations)

C. Garbage collection of objects (e.g. arena allocators etc)

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Sat, 09 Jun 2012 16:09:25 +0200
Subject: Re: Ada.Storage_IO. Anyone use it?
Newsgroups: comp.lang.ada

Ada.Storage_IO is as useful or useless as the other traditional *_IO packages. For an example of usefulness, I can declare one or any number of Buffer_Type objects to put items on hold that were just read form a file ("ungetc"), or will be written to another file when some condition becomes true. Or, more generally, I can use Ada.Storage_IO for in-memory buffering *without- leaving the traditional IO framework. Like when sorting with three "tapes".

The formal Element_Type is definite, but discriminated records might still work just fine, with defaults. A wrapped array of bytes with implementation-defined Positive'Last being the maximum number of elements seems an unnatural example for use with traditional *_IO packages.

For other types such as database records, or anything that isn't a huge in-memory representation of the universe and everything, Ada.Storage_IO should be fine.

```
type Count is range 0 .. 400;
type List_Of_Things is array (Count range
<>) of T;
type Item_to_Store (Length: Count := 42) is
record
   Things : List_of_Things (1 .. Length);
end record;
```

There are compilers that do have a working implementation of Ada.Storage_IO.

Practicalities of Ada for desktop applications

From: wrp <i3text@gmail.com>
Date: Fri, 8 Jun 2012 13:48:40 -0700
Subject: Practicalities of Ada for app development
Newsgroups: comp.lang.ada

What would you say about using Ada for developing desktop applications? I like what I have read in descriptions of the language, but I have concerns about the practicality of using it.

I develop small to medium sized tools. Tasks are mostly limited to text processing, database management, and simple visualization. I have no need for high precision calculation or support for distributed, concurrent, or real-time control. You can picture for example the size and scope of the traditional Unix utilities. Sometimes I build something larger with a GUI.

For considerations of size and performance, I'm planning to shift from scripting to a compiled language. I develop on Linux but also target Windows and OS X. After trying to consider everything available, I feel that the only serious candidates for me are C and Ada.

1. To begin with, I've heard it said that Ada, designed for embedded system building, is simply not suited to apps for a general computing platform. That's rather vague and I haven't seen any detailed justification of that claim. What would you say?

2. I don't need tools to be free, but they have to be affordable to a small shop. If I'm targeting x86-64 (and possibly ARM) is GNAT the only compiler option I have?

3. How about the quality and availablity of supporting tools like debuggers and profilers?

4. How about production quality, open source libraries for things like Unicode support, sockets, network communication, GUIs, etc? The stuff at www.dmitry-kazakov.de looks good, but most of what I have seen online is from the 1980s and pretty rough to begin with.

5. How well is incremental development supported? I'm thinking of things like modular compiling and speed of the edit-compile-test cycle.

6. Size and latency are sometimes an issue. I've heard that since GNAT is oriented to building larger systems, the executables it produces are comparatively bulky. What is the situation relative to C in that regard?

7. What advanced tutorial material is there for using Ada in this way? Say that I have Norman Cohen's _Ada as a Second Language_. What more advanced material is available on subjects other than concurrency, distributed processing, or real-time systems? On a related note, what projects would you recommend looking at as examples of great code?

From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>
Date: Fri, 08 Jun 2012 14:35:55 -0700
Subject: Re: Practicalities of Ada for app development
Newsgroups: comp.lang.ada

"1." I would say everything you've heard is wrong. Ada is a general-purpose language that has been used successfully in every application domain. Ada is the language of choice for S/W that must be correct. Since I want all my S/W to be

correct, I always use Ada. Examples of small Ada applications I've written include one to choose the signature (like the one at the end of this message), one to choose the sound to be played next time I log in (next time it will say, "What is your favorite color?"), and one to display a quote in a dialog window (today's is, "If you think you got a nasty taunting this time, you ain't heard nothing yet!"). There seems to be a theme there.

A larger application is the Mine Detector game: http://pragmada.x10hosting.com/mindet.html

At work we have a large, concurrent, distributed, soft-real-time, web-accessible call-center application, but large parts of it do DB access and things that would be common in the kind of applications you want to make.

“2.” RR Software's Janus/Ada (for Windows) is reasonably priced. You might also look at Atego. Note that most compiler are for Ada 95. Only 3 of 7 compilers I'm aware of support the entire language in the current standard (published in 2007). GNAT is the only compiler I'm aware of that supports features from the next standard (hopefully published this year).

Ada 95 is a very good language, so that might not be a concern. (Even Ada 83 is a better language than most of the competition.)

“3.” One nice thing about Ada is not needing to use a debugger.

“4.” There are plenty of libraries available. You can find many through adaic.org.

“5.” Unlike C, Ada has modules ("packages"). with them, stubs, and separate compilation, incremental development is supported well.

“6.” As I've said, everything you've heard is wrong. Equivalent programs in Ada and C create executables of about the same size using gcc (the key word is "equivalent"). Robert Dewar of AdaCore claims to have a collection of equivalent Ada and C programs that produce identical object code using gcc.

“7.” Cohen is a pretty good book; I'm not sure that you need anything else. Note that even small applications can sometimes benefit from concurrency. Since Ada tasking is high-level and safe, it would a mistake not to learn about it so you can use it when warranted. Barne's book is a good choice. You might want to look at "Ada Distilled" by Richard Riehle. Again, adaic.org has a list of texts and tutorials.

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 8 Jun 2012 17:40:58 -0700
Subject: Re: Practicalities of Ada for app development
Newsgroups: comp.lang.ada

“One nice thing about Ada is not needing to use a debugger.”

I don't get this comment. Ada is a lot nicer than some languages at preventing you from making certain kinds of mistakes, but no language is able to prevent logic errors and certain dumb typos. And if you make this kind of error and the program doesn't work, just the fact of its being written in Ada doesn't help you much. There may less need for a debugger because Ada will prevent certain types of errors and things like constraint checks will catch some others that would cause havoc in C.

But it can't catch everything, and even if it does find an index that's out of range it won't tell you why the index was out of range. I mean, Ada is a much better language than certain others for writing correct code, but this seems like a gross overstatement.

Or maybe I'm just being parochial here because I designed and wrote the Ada debugger that comes with Irvine Compiler's product. And yes, I do use it to help track down problems. (Including errors in the debugger itself.)

From: BrianG <me@null.email>
Date: Sat, 09 Jun 2012 00:30:49 -0400
Subject: Re: Practicalities of Ada for app development
Newsgroups: comp.lang.ada

“1.” I've used GNAT to build small "Unix-like" utilities (simple to moderate processing, standard-in to standard-out filters, …) for around 18 years. Mostly for DOS/Windows, but I used some, and created some, on Linux. Mostly just whatever I need.

“2.” Isn't there still ObjectAda? There's a limited free version that comes with the Barnes Ada 95 book (but not the Ada 2005 book).

I doubt I've ever (yet) used any 2005 features for this type of stuff. I have (long ago) had to back-port some to Ada 83 (old VAX Ada).

“One nice thing about Ada is not needing to use a debugger.”

But it's available - command line gdb or the GPS IDE for GNAT. (I've used it more for confirming code works as I expect than for tracking down bugs).

“6." These stories usually come from creating simple programs without any thought applied to actual "equivalence", or how they're built (if gcc by default uses dynamic libraries and GNAT uses static).

From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>
Date: Fri, 08 Jun 2012 23:38:27 -0700
Subject: Re: Practicalities of Ada for app development
Newsgroups: comp.lang.ada

I haven't used a debugger for a long time. Usually Ada gives an idea of the kind of error and its location, and I can easily figure out what the problem is. In the rare case that that isn't true, it's quicker to stick in a few Put_Lines than to learn to use the debugger again. This is a self-reinforcing situation, of course.

From: Gautier gautier_niouzes@hotmail.com
Date: Fri, 8 Jun 2012 23:55:47 -0700
Subject: Re: Practicalities of Ada for app development
Newsgroups: comp.lang.ada

Since I develop desktop applications for both my job and for hobby, and GUI apps and command-line apps, my answer is: "yes, you can!".

[…]

For 5, I've never seen an Ada development system that was *not-incremental.

It's probably because Ada is modular from day one (really modular, not the hacks with "include"'s).

Dynamic accessibility

From: sbelmont700@gmail.com
Date: Wed, 13 Jun 2012 14:31:29 -0700
Subject: Dynamic accessibility
Newsgroups: comp.lang.ada

Does anyone have any insight or historical perspective as to why it is that access parameters carry a dynamic lifetime along with them, whereas access discriminants do not? I cannot think of a good reason why you would want to try and explicitly typecast an access parameter anyway, so it would seem easier on everyone had parameters been defined statically as discriminants are (i.e. lifetime as being declared inside the subprogram, so that it is checked by the compiler and forbids all attempts at typecasting).

On the other hand, if there is a good reason for doing it, then it would seem appropriate that one would need the same ability for access discriminants as well; i.e. carry along the dynamic lifetime so that someone could explicitly typecast it and save it somewhere else, exactly like an access parameter.

Is there some sort of esoteric accessibility conundrum that requires the rules be like this, or is it a judgment call? Was it just that the implementation of discriminants would be more costly than that of parameters? Was the intention to provide a mechanism for both, so that a programmer could choose either way? Or is it just that the lack of out parameters for functions and inability to dispatch on named access types required a backdoor in case an unlucky programmer was forced into an access parameter, but needed to get back the 'real' type of controlling operand?

Thanks for any opinions, rants, or special secrets anyone might know.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 21 Jun 2012 13:43:06 -0500
Subject: Re: Dynamic accessibility
Newsgroups: comp.lang.ada

Well, this comes up from time-to-time -- it surely would be better if access discriminants had dynamic accessibility. However, the last time the idea was brought up, it was determined that dynamic accessibility would not work at all for discriminants. I forget the details, sorry, but my recollection was that it was impossible to implement.

BTW, there is concern that the dynamic accessibility of Ada 2005 and Ada 2012 is in fact unimplementable as well. (AdaCore has not yet managed to do so.) The main problem is that we can't quite imagine what the alternative is, so it might have to be done in a very expensive manner.

Thus is good that you can avoid dynamic accessibility by avoiding access parameters; use "aliased in out" in Ada 2012 instead and you get the *right accessibility. ("in out" acts as local; "aliased in out" acts as if the accessibility is the point of the call -- this sounds hardly different, but it makes a huge difference in practice, as you cannot return the first but you can return the second.)

Note that dynamic accessibility is always a bad idea, in that it provides "tripping hazard" -- you might get an exception from a few calls, but not others. It's especially bad as calls from unit tests most likely will work (they're not nested) while ones in actual programs might (calls in nested subprograms are much more common).

I've become convinced that the entire idea of accessibility checks isn't worth the headaches (both in language definition and in practice). I virtually always use 'Unchecked_Access in my code, so the net effect is that I pay overhead for accessibility checks, but they never actually have any effect (positive or negative). 'Access will almost always fail on a parameter, so it doesn't even make sense to try to use it -- and it's very rare to have anything that is not encapsulated (the only time 'Access can be used.

From: sbelmont700@gmail.com
Date: Thu, 21 Jun 2012 17:43:23 -0700
Subject: Re: Dynamic accessibility
Newsgroups: comp.lang.ada

["tripping hazard"]

This was my real concern with the access parameter accessibility; the exception depends entirely on what the client passes in (though the rumor in Ada 2012 is that there exists a mechanism to compare accessibility levels, so that one might be able to conditionally typecast an access parameter…?). It would seem a named type is preferable to an access parameter in any case in which assignment was necessary, especially in 2012 where there is not the 'in' parameter restriction for functions. I'm sure there is an example I cannot think of, but what are the legitimate reasons someone would want to pass an access parameter and have occasion to cast it? It seems backwards to provide a mechanism for ensuring assignment does not happen, and then implementing a workaround to allow it.

As always Mr. Brukardt, your responses are insightful and greatly appreciated; thank you for your continued help and support.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Jun 2012 15:03:35 -0500
Subject: Re: Dynamic accessibility
Newsgroups: comp.lang.ada

Membership on access types in Ada 2012 includes the accessibility check. So you can write a precondition:

```
Pre => Param in My_Access_Type
```

which will fail if the accessibility check would fail.

"… what are the legitimate reasons someone would want to pass an access parameter and have occasion to cast it?"

I don't think there are any in Ada 2012. In Ada 95, you sometimes had to do that as a replacement for the missing "in out" parameter for a tagged object. Usually the trick was to "strip off" the accessibility check:

```
Param.all'Unchecked_Access
```

as you would have had to if the parameter was an "in out" parameter in a procedure. Note that "aliased" parameters help a bit here (they're guaranteed to live as long as the result of the function call, which is a bit longer than a "normal" parameter).

SPARK and aggregates

From: Ben Hocking <benjaminhocking@gmail.com>
Date: Thu, 14 Jun 2012 05:43:42 -0700
Subject: Having problem with SPARK Ada complaining that 'No EXPRESSION can start with reserved word "OTHERS".'
Newsgroups: comp.lang.ada

The following is a simple test program to demonstrate the issue.

test.ads:

```
package Test is
   type QState is (Off, On, Both);
   subtype Z4 is Integer range 0 .. 3;
   -- 0, 1, 2, or 3
   type Ensemble is array (Z4) of QState;
   procedure Foo;
end Test;
```

test.adb:

```
package body Test is
   procedure Foo
   is
      SomeQState : Ensemble := (others =>
Off);
   begin
      null;
   end Foo;
end Test;
```

When I use SPARK->Examine File (from GPS), I get the following message:

4:33 Syntax Error No EXPRESSION can start with reserved word "OTHERS".

I get the same error if I type "spark test.adb" from the terminal window (I'm on Mac OS X 10.6.8).

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Thu, 14 Jun 2012 14:37:43 +0100
Subject: Re: Having problem with SPARK Ada complaining that 'No EXPRESSION can start with reserved word "OTHERS".'
Newsgroups: comp.lang.ada

In SPARK all aggregates must be qualified with the type:

```
SomeQState : Ensemble :=
          Ensemble'(others => Off);
```

This isn't particularly clear in any of the documentation, but see Section 4.3 of the SPARK LRM or page 110 of the book (High Integrity Software by John Barnes).

From: roderick.chapman@googlemail.com
Date: Thu, 14 Jun 2012 14:48:24 -0700
Subject: Re: Having problem with SPARK Ada complaining that 'No EXPRESSION can start with reserved word "OTHERS".'
Newsgroups: comp.lang.ada

In the documents that come with the GPL edition, see section 4.3 of SPARK_LRM.pdf - the rules are clearly stated there.

SPARK substitution rules

From: Ben Hocking <benjaminhocking@gmail.com>
Date: Wed, 20 Jun 2012 11:30:53 -0700
Subject: Need help understanding SPARK substitution rules
Newsgroups: comp.lang.ada

I have the following math.ads:

```
package Math is
   function IsPositive (InVal : in Integer)
            return Boolean;
   -- # return (InVal > 0);
   function IsNegative (InVal : in Integer)
            return Boolean;
   --# return (InVal < 0);
   function AlwaysTrue (InVal : in Integer)
            return Boolean;
```

```
   --# return True;
end Math;
```

and math.adb:

```
package body Math is
   function IsPositive (InVal : in Integer)
            return Boolean is
   begin
      return (InVal > 0);
   end IsPositive;
   function IsNegative (InVal : in Integer)
            return Boolean is
   begin
      return (InVal < 0);
   end IsNegative;
   function AlwaysTrue (InVal : in Integer)
            return Boolean is
   begin
      --# check not (InVal > 0) -> InVal <= 0;
      --# check not IsPositive(InVal) -> InVal <=
            0;
      --# check not (InVal < 0) -> InVal >= 0;
      --# check not IsNegative(InVal) -> InVal
            >= 0;
      return (not IsPositive(InVal) or not
            IsNegative(InVal));
   end AlwaysTrue;
end Math;
```

The statement "check not (InVal > 0) -> InVal <= 0" is proven easily enough, but the next line ("check not IsPositive(InVal) -> InVal <= 0") is not, even though it's functionally equivalent. If I put the following line in math.rlu:

math_ispositive: ispositive(inval) may_be_replaced_by (inval > 0)

It now can prove it, and if I do the same thing with is negative, the whole thing proves out (with the "return True" component being proven by ViCToR).

However, this seems like a very fragile way of assuring code correctness, so it reeks from "code smell". What is a better way of achieving this?

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Thu, 21 Jun 2012 09:47:20 +0100
Subject: Re: Need help understanding SPARK substitution rules
Newsgroups: comp.lang.ada

What you need is for the Examiner to include the return expressions in the hypotheses - it does (now) do this, but only for VCs generated for code that follows the function call. (See Proof Manual Section 8.4.9 - bullet point 4)

So if you change your code to

```
function AlwaysTrue (InVal : in Integer)
return Boolean is
   Result : Boolean;
begin
   Result := (not IsPositive(InVal) or not
         IsNegative(InVal));
   --# check not (InVal > 0) -> InVal <= 0;
   --# check not IsPositive(InVal) -> InVal <=
         0;
   --# check not (InVal < 0) -> InVal >= 0;
   --# check not IsNegative(InVal) -> InVal
         >= 0;
   return Result;
end AlwaysTrue;
```

then the VCs generated for the check annotations include hypotheses such as:

H7: ispositive(inval) <-> (inval > 0) .

Now the only VC left unproven by the Simplifier is the last one (which you note is proved by Victor).

For those of us who (so far) find Victor (I loathe that casing they've adopted) not practicable for anything other than trivial code snippets, the one remaining VC can be proved by the rule:

boolean(1): not A or not B may_be_deduced_from [A <-> X > 0, B <-> X < 0] .

which has the merit of being universally true and avoiding any "code smell".

Furthermore this rule can then be validated by proving the corresponding VC:

H1: a <-> x > 0 .

H2: b <-> x < 0

->

C1: not a or not b .

(There are hints that this approach to validating rules will be described in the next version of the SPARK book).

Dimension checking with GNAT

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Sat, 30 Jun 2012 22:53:57 -0700
Subject: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

I see that GNAT now has added packages System.Dim.- for dimension checking. However I cannot find any other documentation about these packages, neither in GNAT_UG nor in GNAT_RM.

Is there any?

From: Nasser M. Abbasi <nma@12000.org>
Date: Sun, 01 Jul 2012 01:25:21 -0500
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

"Performing Dimensionality Analysis in GNAT"[1,2] "Documentation for GNAT dimensionality checking system"[3]

The ads file for this package is s-sim.ads in the sources.

[1] http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/Performing-Dimensionality-Analysis-in-GNAT.html

[2] http://docs.adacore.com/gnat-unw-docs/html/gnat_ugn_28.html

[3] http://www.mail-archive.com/gcc-patches@gcc.gnu.org/msg26036.html

From: yogeshwarsing@gmx.com
Date: Sun, 1 Jul 2012 03:44:19 -0700
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

This feature is rather limited though. For example one cannot do conversions. If you have to work with an "irregular" unit, you will have to do the conversion yourself e.g.

```
with System.Dim.MKS;
use System.Dim.Mks;
with System.Dim.Mks_IO;
use System.Dim.Mks_IO;
with Text_IO; use Text_IO;
procedure Free_Fall3 is
   subtype Acceleration is Mks_Type
   with Dimension => ("m/s^2", Meter => 1,
         Second => -2, others => 0);
   G : constant acceleration := 127137.6-
         km/(hour ** 2) ;
   T : Time := 10.0/3600.0- hour;
   Distance : length;
begin
   Put ("Gravitational constant: ");
   Put (G, Aft => 2, Exp => 0); Put_Line ("");
   Put ("Time: ");
   Put (T, fore => 4, Aft => 4, Exp => 0);
   Put_Line ("");
   Distance := 0.5- G * T ** 2;
   Put ("distance travelled in 10 seconds (or
         10/3600 hour) of free fall ");
   Put (Distance, fore => 4, Aft => 4,
         Exp => 0);
   Put_Line ("");
end Free_Fall3;
```

You will still get the outputs in the default MKS units. So basically, it is just checking if your dimensions are right.

And I do not think that you are able to deal with record constructs such as

```
type Motion_Parameter is record
   Radius: Length := 0.0- m;
   Speed : Velocity := 0.0- m/s;
   Time_Step    : Time :=  0.0- s;
   Number_Of_Revolutions : Float := 0.0;
end record;
```

since Motion Parameter will need to have a specific dimension in terms of MKS and having different MKS units in the record structure makes this impossible.

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Sun, 1 Jul 2012 08:18:37 -0700

Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

Thanks for the chapter - I just overlooked it.

The documentation, though, is rather terse. It doesn't tell you the syntax of the aspect clauses nor anything else than the most basic things.

All in all it looks promising - it's a very clever way to use the new aspects.

What I found out:

```
G : constant acceleration :=
      127137.6- km/(hour ** 2);
Put (Sqrt (G), Aft => 2, Exp => 0);
Put_Line ("");
```

yields 3.13 m**(1/2).s**(-1), so it handles fractional powers (at least 1/2).

Math functions except sqrt need dimensionless parameters. However, sin (t, t0) should be allowed as long as t and t0 have the same dimension. It is not. Also arctan (x, y) is only allowed for dimensionless parameters.

We'll see how it turns out when applied to more complicated formulae, especially with those where parts have fractional dimensions (in SI, there are no items with fractional units, but intermediate ones do).

CGS cannot do without fractions.

Perhaps with some refinement, this could be a theme for Ada 2020 :-)

PS: Anyone interested in dimensional arithmetics in Ada should have a look at:

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

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Mon, 2 Jul 2012 13:42:04 -0700
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

I may be off track here. That is I thought I understood aspect syntax until I browsed to the definition of System.Dim.Mks.Mks_Type type.

```
type Mks_Type is new Long_Long_Float
  with
   Dimension_System => (
    (Unit_Name => Meter,    Unit_Symbol
          => 'm',   Dim_Symbol => 'L'),
    (Unit_Name => Kilogram, Unit_Symbol
          => "kg",  Dim_Symbol => 'M'),
    (Unit_Name => Second,   Unit_Symbol
          => 's',   Dim_Symbol => 'T'),
    (Unit_Name => Ampere,   Unit_Symbol
          => 'A',   Dim_Symbol => 'I'),
    (Unit_Name => Kelvin,   Unit_Symbol
          => 'K',   Dim_Symbol => "Theta"),
    (Unit_Name => Mole,     Unit_Symbol
          => "mol", Dim_Symbol => 'N'),
    (Unit_Name => Candela,  Unit_Symbol
          => "cd",  Dim_Symbol => 'J'));
```

Can anyone point me to the ARM 2012 paragraph(s) which supports this syntax in general.

From: Adam Beneschan <adam@irvine.com>
Date: Mon, 2 Jul 2012 13:58:41 -0700
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

13.1.1(38): "Implementations may support implementation-defined aspects. The aspect specification for an implementation-defined aspect may use an implementation-defined syntax for the aspect definition, and may follow implementation-defined legality and semantics rules."

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Fri, 6 Jul 2012 03:47:58 -0700
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

I would like to invite all of you interested in dimensional algebra to a lively discussion about the findings in packages provided by GNAT.

I played around a bit and here are mine.

GNAT Dimension_System Findings

Very clever use of the new aspect facility.

Should be considered for standardisation in next Ada generation.

Currently very poorly documented.

There are a few minor problems and some (in my honest opinion) more severe ones in output.

The following is based on the (Gaussian) CGS system.

```
type CGS_Gauss is new Long_Long_Float
   with Dimension_System => ((Centimeter,
        "cm"),(Gram      , 'g'), (Second    , 's'));
package CGS_Gauss_IO is new
         System.Dim.Float_IO (CGS_Gauss);
 use CGS_Gauss_IO;
 subtype Length is CGS_Gauss
  with Dimension => ("cm",
                     Centimeter => 1,
                     others     => 0);
 subtype Mass is CGS_Gauss
  with Dimension => ("g",
                     Gram   => 1,
                     others => 0);
 subtype Time is CGS_Gauss
  with Dimension => ("s",
                     Second => 1,
                     others => 0);


 cm: constant Length := 1.0;
 g : constant Mass   := 1.0;
 s : constant Time   := 1.0;
```

0. The syntax of aspects Dimension_System and Dimension is not documented.

It might seem obvious, but documentation is needed nevertheless.

In Dimension_System, up to 7 base dimensions may be defined (more lead to a compilation error, less are tolerated).

1. How to define objects for dimensions without name?

Imagine you have some charge, but not defined a subtype Charge.

```
Q: CGS_Gauss := 40.0*cm**(3/2)*g**(1/2)/s;
-- ???
```

This fails with

dimensions mismatch in object declaration

object type is dimensionless

object expression has dimensions (3/2, 1/2, -1)

It's no problem to define the subtype

```
subtype Charge is CGS_Gauss
 with Dimension => ("esu",
                    Centimeter => 3/2,
                    Gram       => 1/2,
                    Second     => -1);
```

and then write

```
Q: Charge := 40.0- cm**(3/2)*g**(1/2)/s;
```

but it is often the case that some intermediate value has to be stored with an unnamed dimension. Very inconvenient if you have to locally define a subtype for this.

** Dimension should be taken from the initial expression! **

2. Obviously GNAT can handle fractional dimensions.

This is very comfortable.

```
Q: Charge := 40.0- cm**(3/2)*g**(1/2)/s;
R: Length := 10.0- cm;
Put (Q**2/R**2   , Aft => 2, Exp => 0);
New_Line;
Put (Q**2/R**2   , Aft => 2, Exp => 0,
          Symbols => "dyn");
New_Line;
Put ((Q/R)**(5/7), Aft => 2, Exp => 0);
New_Line;
16.00 cm.g.s**(-2)
 2.69 cm**(5/14).g**(5/14).s**(-5/7)
```

However, I cannot find where

```
function (Left: Dim_Type; Right: Rational)
return Dim_Type;
```

is defined, let alone the type Rational.

The definition of Rational is flawed.

Don't write

```
(8.0*cm)**(1/3+2/3)
```

this has the value 1. This, however, is correct:

```
(8.0*cm)**((1+2)/(5-2)) = 8.0*cm
```

(Admittedly, who would write such nonsense? Dimension checking is only done for static expressions.)

Ahem - be careful:

```
8.0**(1/3)     = 1
8.0**(1/3)*cm  = 2.0*cm
(8.0*cm)**(1/3) = 2.0*cm**(1/3)
```

You need a dimensioned value to give the correct result:

```
One: constant CGS_Gauss := 1.0;          -
- dimension 1, i.e. "dimensionless"

8.0**(1/3)*One = 2.0
```

(In SI, results never have fractional dimensions, but intermediate values may. So this is important also for SI.)

3. System.Dim.Float_IO is only output.

Values are correctly output with their dimension, very nice.

However, the name Float_IO is a lie, there is no dimensional input.

Input would be hard to achieve with the output syntax as produced now because how to determine whether a dimension to read follows a number or not since there is a separating space. We would need a multiplication sign instead.

```
Put (Q**2/R**2, Aft => 2, Exp => 0);
```

results in

```
16.00 cm.g.s**(-2)  <=== How to read this
                         back in?
```

Also the exponent in an output like 1.0 m**2.s could be taken as the floating point number 2.0 when trying to read it in again (in input, missing fore or aft is allowed).

Compare with

```
16.00*cm.g.s**(-2)  <=== This is
                         dimensioned.
```

So checking the input for syntactic and semantic correctness is not easy.

Adding a symbol for the output is not checked! You can write any nonsense.

```
Put (Q**2/R**2, Aft => 2, Exp => 0, Symbols
        => "dyn");
New_Line;
Put (Q**2/R**2, Aft => 2, Exp => 0, Symbols
        => "m/s");
New_Line;
16.00dyn     <=== space missing (dyn
                  undefined)!
16.00m/s     <=== nonsense!
R: Length := 10.0- cm;
Put (R, Aft => 2, Exp => 0);
New_Line;
Put (R, Aft => 2, Exp => 0, Symbols =>
        "km");
New_Line;
10.00 cm     <=== OK
10.00km      <=== Surprise, surprise!
```

This behaviour is apt to lead to confusion and program bugs!

4. Mathematics is included!

```
package Math is new Ada.Numerics.
        Generic_Elementary_Functions
        (CGS_Gauss);
  use Math;
```

Any function requires dimensionless parameters and returns a dimensionless result, except Sqrt, which correctly calculates the root of the dimension.

However:

```
function Sin (X, Cycle : Float_Type'Base)
return Float_Type'Base;
```

(Cos, Tan, Cot) should allow dimensioned parameters when both have the same dimension.

```
function Arcsin (X, Cycle : Float_Type'Base)
return Float_Type'Base;
```

(Arccos) should request X dimensionless and the return value should be dimensioned like Cycle.

```
function Arctan (Y: Float_Type'Base;
   X: Float_Type'Base := 1.0[;
   Cycle : Float_Type'Base])

return Float_Type'Base;
```

(Arccot) should allow X and Y with the same dimension [and be dimensioned like Cycle].

Any thoughts, other nice features or awkward shortcomings?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 8 Jul 2012 13:37:16 +0200
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

[…] Actually, static-only dimension checks have almost no use for engineering. Scientific computations might be different, however require more than just dimensioned scalars, e.g. dimensioned vectors, matrices etc. I think they just choose a units system and don't care about dimension analysis.

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: Sun, 08 Jul 2012 20:47:07 +0200
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

Some of us are patiently waiting for Ludovic et al. to merge the relevant patches into the Debian GNAT packages.

I find it interesting that AdaCore seem to have found a general solution to the problem, but for the moment I manage fine with a slightly modified version of Macks and statically generated checks.

From: Martin Dowie <martin@thedowies.com>
Date: Mon, 9 Jul 2012 01:20:53 -0700
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

I really like the idea of attempting this analysis via aspects. I remember you had a few examples of why Tucker's attempt via signature packages came up short, are many of the same issues present in the aspect solution?

From: AdaMagica <christ-usch.grein@t-online.de>
Date: Mon, 9 Jul 2012 04:19:39 -0700
Subject: Re: GNAT and Dimension Checking
Newsgroups: comp.lang.ada

See Ada 95 Issue 324 for Tucker's packages and discussion.

No, all those problems with the combinatorial explosion of functions are gone with this really ingenious use of aspects.

The method keeps a record of the (possibly fractional) dimension of each item at compile-time just like my packages[1] do at run-time. It's very easy to use and dimensioned literals look nice.

[1] http://www.christ-usch-grein.homepage.t-online.de/Ada/SI.html

SPARK GPL 2012 and generic

From: hugues@soekris-1.pinkyoogx.org
Date: 08 Jul 2012 10:17:10
Subject: SPARK GPL 2012 and generic, any hint?
Newsgroups: comp.lang.ada

I tried SPARK GPL 2012 with a small example for generic, but I keep getting syntax error, do I miss something obvious?

The documentation lacks a complete example that can be processed, all I could find was limited code snippet :(

Here is the code source (code taken from SPARK GPL 2012 documentation, I simply added package stuff around)

```
package Test is
  generic
    type T1 is range <>;
    type T2 is range <>;
    --# check T1'Last- T1'Last <= T2'Last and
    --#       T1'First- T1'First <= T2'Last and
    --#       T1'First- T1'First >= T2'First;
  function Square (X : T1) return T2;
  --# return R => R = T2 (X- X);
end Test;
```

and the result for the Examiner

Examiner GPL 2012

****

DATE : 08-JUL-2012 12:22:29.763
generic^

**- Syntax Error : reserved word "IS" cannot be followed by reserved word "GENERIC" here.

--- Warning :430: SLI generation abandoned owing to syntax or semantic errors or multiple units in a single source file.

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Sun, 8 Jul 2012 16:31:22 +0100
Subject: Re: SPARK GPL 2012 and generic, any hint?
Newsgroups: comp.lang.ada

The instantation check isn't supported yet, and the subprogram must be at library level.

The following code examines OK:

```
generic
   type T1 is range <>;
   type T2 is range <>;
   function Square (X : T1) return T2;
  --# return R => R = T2 (X- X);
```

Subroutine signatures duplicated in interfaces and bodies

From: Nasser M. Abbasi <nma@12000.org>
Date: Sat, 07 Jul 2012 18:08:53 -0500
Subject: on the need to duplicate code for procedure signature in both body and interface files
Newsgroups: comp.lang.ada

Basic question from an Ada newbie.

One thing that always bothered me in Ada is the need to physically duplicate the code that represents the API of the package procedures and functions in both the interface and the body files (.ads and .adb).

I do not like duplicating code at all. Even if it is only for the signature of the API, and even though the Ada compiler will catch any difference (assuming one changes the .ads and forget to update the .adb for example).

Was there no other alternatives to avoid this situation when Ada was originally designed in order to keep the same good concept of separating the interface from the body, but somehow at the same time, eliminate the need to duplicate by hand the API definition code in 2 separate places?

I am using GNAT. I Assume this is the same in other Ada implementations.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 08 Jul 2012 07:21:07 +0300
Subject: Re: on the need to duplicate code for procedure signature in both body and interface files
Newsgroups: comp.lang.ada

[…]

That is a language property that applies to the source-code files. If you use some kind of CASE tool or IDE, it can hide the duplication by synchronizing the text in the spec and body. For example, I think GPS/GNAT has the ability to generate a dummy body (.adb) from a given spec (.ads) (but not to maintain the connection if the spec changes). Possibly the Ada mode of emacs can do that too.

Several years ago I used an Ada CASE tool called HoodNICE, based on the HOOD design method. In this tool, one wrote the Ada code once, and the tool generated the spec and body files. However, the design method was based on packages as "objects", and dependencies were basically managed on the object level *without- separation of spec and body, so one had to take some special ugly action to show that "object" A depended on "object" B only because the *body- of package A needed a "with" clause for package B, while the *spec- of A did *not* need such a clause.

(To be fair, the HOOD design method tried to enforce a strict layering with no mutual dependencies through bodies. But this is a very limiting rule, and we violated it in most projects.)

[…]

The important reason for "duplicating" the code (signature) is that one needs some way to show which body belongs to which declaration when the subprogram names are overloaded (same name, different signature).

Duplicating the text that defines the signature is IMO the simplest and most readable way to show this connection, when the declaration and body are in different files. in a sense, there is no more "duplication" here than when we "duplicate" the subprogram name and parameter names in the calls to the subprograms.

Keeping the spec and body in the same file would mess up "make" logic based on file time-stamps. Changing the body code should normally not force recompilation of clients of the spec.

[…]

Given the copy/paste abilities of good text editors, I don't see this as a problem at all. In my spec files, most of the text is comments (ooh how I hate that misleading word!) that describe what the stuff means and how it works; the actual Ada code is a small fraction, and of this code, only the subprogram signatures are duplicated in the body. (Of course there are comments in the body, too, but on different issues.)

No doubt one could devise an Ada compilation system, similar to the HoodNICE method, in which only body files exist, with some mark-up to show which parts of these files are "public" (belong in the spec). A dedicated tool could then generate the spec files, and also avoid updating the time-stamp of the spec when the contents of the file do not change. But as I said, I don't feel that this (apparent) duplication is a problem.

[…]

I wonder how the Rational Ada environment worked, on this issue. I'm sure someone on this group remembers.

From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>
Date: Sat, 07 Jul 2012 23:09:52 -0700
Subject: Re: on the need to duplicate code for procedure signature in both body and interface files
Newsgroups: comp.lang.ada

[…]

Consider the following:

```
package P is
   function F (I : Integer) return Integer;
   type Integer_List is array (Integer range
         <>) of Integer;
   function F return Integer_List;
end P;
```

How are you going to write the body of P so the compiler knows which body implements which function F without repeating the parameter list and return type profile, which is what distinguishes these 2 overloaded functions?

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 08 Jul 2012 03:38:50 -0400
Subject: Re: on the need to duplicate code for procedure signature in both body and interface files
Newsgroups: comp.lang.ada

“I do not like duplicating code at all.”

Why?

“… even though the Ada compiler will catch any difference”

That covers the main objection I'm aware of; that duplicated code quickly becomes wrong.

From: Britt <britt.snodgrass@gmail.com>
Date: Sun, 8 Jul 2012 18:48:09 -0700
Subject: Re: on the need to duplicate code for procedure signature in both body and interface files
Newsgroups: comp.lang.ada

This topic was recently discussed in the "Ada Programming Language" LinkedIn group in a thread titled "Imaginary proposal for the next Ada standard: Ada compilers will automatically generate Package Specification from Package Body".

The discussion started as a poll and most respondents (including me) strongly dislike the idea.

Edward Colbert just posted a nice summary there.

Thin/medium/thick bindings

From: Nasser M. Abbasi <nma@12000.org>
Date: Thu, 02 Aug 2012 03:30:27 -0500
Subject: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

I was trying to make a small Ada openGL program using the openGL bindings in the GLOBE3D packages downloaded from

http://globe3d.sourceforge.net/

I was simply trying to do the same as I did with Fortran openGL, where I took a C example from the openGL red book, and did 1-1translation of the API and build it.

But the first thing I noticed once I started looking more at the Ada binding, is that with the Ada openGL binding the C function names have been changed a little in the Ada binding.

This is not really good. Unless I am overlooking something or looking at the wrong files.

The binding in a thin binding, and hence the function names in the Ada side should be 100% the same as the C names. This makes it very easy to port C open GL to Ada, and to use the C openGL books since the API has the same calls.

For example, looking at the file binding/gl.ads that is part of the GLOBE3D binding, I see the following:

```
pragma Import (Stdcall, GL_Begin,
"glBegin");
pragma Import (Stdcall, GL_End, "glEnd");
```

Why the name was changed? This should be

```
pragma Import (Stdcall, glBegin, "glBegin");
pragma Import (Stdcall, glEnd, "glEnd");
```

And the 'gl' was removed from all the function names from the rest of the calls. For example

```
pragma Import (Stdcall, Vertex2d,
"glVertex2d");
pragma Import (Stdcall, Vertex2f,
"glVertex2f");
```

These should be

```
pragma Import (Stdcall, glVertex2d,
"glVertex2d");
pragma Import (Stdcall, glVertex2f,
"glVertex2f");
```

It does not matter that these are already defined in 'gl' package. I can choose to write

```
with gl;
gl.glVertex2d(…)
```

or

```
with gl; use gl;
glVertex2d(…)
```

The point is, the binding should be the same name as the C name.

My questions:

How hard is it to fix all this to make the Ada openGL binding names consistent with the C names? I can help in doing these changes to all the files. (should not be that hard I would think?, unless there are other hidden issues I am not seeing).

Also, on a side note, I noticed that glutInit() is on the body of the glut packages, but it is not in the specification of the glut package for some reason.

And on a final note, even though Ada is not case sensitive like C, I think these binding should also be written in mixed case in the same way as the C standard shows them. Even though it makes no difference on the Ada side of things, it makes the Ada code using the openGL API look the same as the C code, which means it is easier for read since that is how the OPENGL standard looks like.

From: Egil Høvik <egilhovik@hotmail.com>
Date: Thu, 2 Aug 2012 01:50:19 -0700
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

So no, the names in an Ada-binding does not have to exactly match the C version. As for GL_Begin and GL_End, they should have been called just Begin and End, but those are reserved words in Ada.

From: Nasser M. Abbasi <nma@12000.org>
Date: Thu, 02 Aug 2012 04:14:15 -0500
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

I know they do NOT HAVE TO be the same as C. That is my point.

They do NOT have to, yes, but it is _better_ if they DO. Why not keep the names the same?

It will make it easier to program this in Ada if the binding is the same.

[…]

NO, they should have been called the same as C. glBegin and glEnd. No need to make up new names. The names are already defined. Why chop off anything.

Btw, I just downloaded at the other Ada openGL now

http://adaopengl.sourceforge.net/downloads.php

and I see that the binding there is the SAME as C binding, which is good. So I am looking at it now.

Here is an example from the file adaopengl\opengl.ads in the above zip file:

…

```
pragma Import (C, glBegin, "glBegin");
pragma Import (C, glEnd, "glEnd");
pragma Import (C, glVertex2d,
  "glVertex2d");
pragma Import (C, glVertex2f,
  "glVertex2f");
```

You can see, the Ada binding above matchs the same as the C API. Even with the mixed case on the Ada side.

I would have liked to use the GLOBE3D openGL binding, because my understanding it is more updated than the David Holm one which was last updated in 2002. But it is more important for me to use a binding which has the same exact names as C as I use C book to learn openGL and I like the code to look the same.

This was the case with the Fortran binding as well, it have the same exact API naming as C. So, I see no reason at all to change the name of the functions, even though I know it is allowed.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Thu, 02 Aug 2012 13:19:56 +0300
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

“They do NOT have to, yes, but it is _better_ if they DO.”

Subjective opinion and matter of taste.

FWIW, I prefer the Ada-style names. The changes (at least in the examples you showed) are so systematic and simple that it is easy to translate in one's mind, when necessary.

“Why not keep the names the same?”

The C-style names are needlessly long and tedious. The "gl" prefix is necessary (well, almost) in C, for name-space reasons, but unnecessary in Ada. If you like, in your Ada code you can include the prefix by using the GL package name to qualify the names. Adding a period to change glVertex2d into GL.Vertex2d is hardly difficult.

IMO, it is neater and less trouble to use the same style of identifiers for my native Ada code and for the bound libraries from other languages. I think that is one of the desirable features of a binding, even a thin binding

From: Nasser M. Abbasi <nma@12000.org>
Date: Thu, 02 Aug 2012 05:46:35 -0500

Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

[…]

Why not keep things standard? The standard is there. No reason to change the names just because you do not like this name and I do not like that name.

When one looks at C code, all the calls are the same everywhere.

"The C-style names are needlessly long and tedious."

It does not matter really. It is the standard. It makes the code easier to understand if the same names are used in Ada and C and Fortran and in any other language. The same exact names.

Anyway, that is what I think. We agree to disagree I guess.

I'll use the 2002 openGL binding to learn a little bit of Ada openGL from, since it matches the C API. So, I am all set now :)

From: Egil Høvik <egilhovik@hotmail.com>
Date: Thu, 2 Aug 2012 04:12:51 -0700
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

"Why not keep things standard? The standard is there."

You seem to be confused. What you call "the standard" is just the C binding. The standard explicitly states that example syntax is in C, and that other languages with better namespace handling and subprogram overloading can do things differently.

Writing GL.Vertex is more readable than GL.glVertex2d. Let the compiler figure out the types of your parameters, and how many, you specify. It will complain if it can't find a match.

Some people complain about Ada being too verbose; In this case, I would say C is more verbose, why should that be a bad thing for Ada?

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Thu, 02 Aug 2012 13:56:03 +0200
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

Perception provides for a valid argument: If books about OpenGL, and other material considered relevant, use the names and example syntax, this creates an expectation. To dispel the power of expectations, you'd need to name a few killer features, features that warrant deviation from what everybody else is perceived to be doing. Or you'd demonstrate, convincingly, that a significantly perceivable number of relevant teams do *not- use the expected names and syntax.

There were two bindings to the OS/2 API. One binding copied IBM's names exactly, as they were used in IBM's documentation, reflecting the names in the C based O-O system: DosXyz123, GpiSomeThing, WinEtcFoo. The other binding made Dos, Gpi, Win, … into packages. Not everyone agreed with the second approach. The argument against package might be stronger in this case because IBM's O-O design was written in C, IIUC, so these were the "real" names.

It there a strong technical argument in favor of using package software instead of names implying packages? Better visibility control? Better compilation performance due to separation? Better change management by modularization?

From: "Vasiliy Molostov" <molostoff@gmail.com>
Date: Thu, 02 Aug 2012 16:01:23 +0400
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

"Some people complain about Ada being too verbose; In this case, I would say C is more verbose, why should that be a bad thing for Ada?"

It seems that you probably right - there is a good reason to keep things convenient to Ada related environment. Perhaps, interface of some sort of C that lie on C naming is indeed a mangled names used to import, and is not going to be human convenient.

BTW, initially GL library from MS had no gl prefix.

Do we need operate mangled names in a high level language?

Which standard we should apply here?

From: "Vasiliy Molostov" <molostoff@gmail.com>
Date: Thu, 02 Aug 2012 16:12:17 +0400
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

Do we need use name prefixes while having high level tools to organize imported procedures in a way we decide by self?

Prefix is a C "necessity", and when imported it then has nothing common with C.

It seems, that a good binding can provide many benefits when imports being organized in an Ada-way, only if you are not going to provide low-level direct 1:1 mapping to what people use in C.

Is there a benefit using 1:1 C in Ada?

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Thu, 2 Aug 2012 12:13:09 +0000
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

[…]

"Making code look the same as C" is … not unambiguously better!

"It will make it easier to program this in Ada if the binding is the same."

No, but I'd agree it will make learning by following C examples a little bit easier. Which is important, up to a point.

Actually programming, and reading and maintaining such a program, is likely to be that much harder, as the names are longer, uglier and therefore less readable.

I confess I didn't watch this year's Tour de France. But I wonder, how many of the competitors used training wheels?

Just a point of view…

There is a long tradition among C programmers, of uglifying names as a poor substitute for properly indicating their type. But there is less than no reason to follow that tradition in Ada, where types and packages can be used to properly distinguish different entities.

So I would agree with Niklas' suggestion : if you must mimic the look of the C programs, substitute '.' for '_' and use qualified names. Ditto where there is any real danger of confusing a gl.Vertex2d with any other type of Vertex2d. But otherwise, the cleanest and simplest style will be best in the long run.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Thu, 02 Aug 2012 11:01:50 -0400
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

""Making code look the same as C" is … not unambiguously better!"

But sticking closely to the C does have the advantage that the binding doesn't need a whole lot of documentation (which must be maintained) -- you can just refer to the C docs. A thick binding requires a lot of documentation.

IMHO, the name changes are so systematic in this case, that it's close enough -- you can still refer to the C docs, and do the trivial translation in your head. And the Ada names really are more readable. I think it's the right choice, especially since the standard actually suggests doing it this way in languages that have a proper module system.

What if you had a system originally written in Ada (with appropriate use of overloading), and you wanted to make a C binding? In that case it would be impossible to make the names the same.

I once wrote an Ada binding to some C code where every C function returned an 'int' as an error code, with 0 meaning "success". I did it in two layers. A thin binding that worked the same way, returning Interfaces.C.int. Then a layer on top of that that turned the error codes into raising an exception. The second layer is such a systematic change that I think it

can still be considered "thin" -- thin enough that the C documentation still makes sense.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 2 Aug 2012 15:59:52 -0500
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

"IMO, it is neater and less trouble to use the same style of identifiers for my native Ada code and for the bound libraries from other languages. I think that is one of the desirable features of a binding, even a thin binding."

Not to mention that automated style-checking tools would reject/modify names in mixed case style. (It's a clear violation of pretty much every Ada style guide I've ever seen.

And personally, whenever I type identifiers, they naturally come out in the Ada style. On the rare occasions when they need to be in some other style (as in specifying link names in interfacing pragmas), I usually have to type them several times.

Ergo, if you want a binding to be useful to the experienced Ada programmer, it has to use identifiers that fit the style of experienced Ada programmers.

IMHO, I do agree with you that a binding that changes the names is not a "thin" binding. "Thin" bindings are unusable; they should only be used for temporary (one-off) code. Anything else needs at least what I call a "medium" binding: routines with similar semantics to the original ones, but an Ada-ized interface with better names, far fewer pointers, and exceptions rather than error codes. (The project that became Claw was originally intended to produce a "medium" binding for Win32. Eventually it morphed into Claw, a classic "thick" binding.)

(I don't want any of my code to *ever-look anything like C -- I want clean, elegant code, not barely intelligible symbols and loads of dangerous pointers.

YMMV. :-)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 2 Aug 2012 16:12:49 -0500
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

Many of us write almost exclusively in Ada, and mixed case C names are a lot harder to understand than properly formatted Ada names. You want our code to be much harder to understand (by us and our colleagues) so that it is easier to read C code.

Sorry, as soon as an Ada programmer has to read C code in order to get their work done, Ada has lost. That programmer would probably have been better off writing the code in C in the first place. The effort of building a binding is not just the binding (that's usually pretty easy), but also the effort of creating/translating examples (and testing them), and preferably, creating Ada-specific documentation as well.

Given that we are forced into a mixed-language world, there's little point in trying to write C in Ada. Either write Ada in Ada or C in C -- and interface *those* larger parts.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 2 Aug 2012 16:18:05 -0500
Subject: Re: questions on Ada openGL binding in the GLOBE3D packages
Newsgroups: comp.lang.ada

"… Then a layer on top of that that turned the error codes into raising an exception. The second layer is such a systematic change that I think it can still be considered "thin" -- thin enough that the C documentation still makes sense."

I think calling that "thin" is dubious. I call the technique "medium" for a lack of a better term, and it has to include normalizing the names, adding appropriate defaults, and using Ada parameter modes appropriately (along with raising exceptions). But I agree that it makes reading C documentation easier than for a true "thick" binding -- the problem being, that no one should be forcing Ada programmers to read C documentation. Cost considerations of course make that necessary sometimes, but even then I would hope that there would at least be some Ada-specific documentation covering common use-cases.

Run-time error for integer division with non-zero remainder

From: Nasser M. Abbasi <nma@12000.org>
Date: Sat, 11 Aug 2012 20:14:42 -0500
Subject: can Ada give run-time error or warning for integer division with non-zero remainder?
Newsgroups: comp.lang.ada

In Ada when dividing 2 integers, the result is an integer with the remainder ignored. […]

Now, suppose I want to know that a division between 2 integers has resulted in nonzero remainder that was thrown away. Maybe because my algorithm is meant to work only for even values and an odd value means there was a bug somewhere and I want to know about it.

Is there any kind of run-time switch to tell it to check for this? I know I can always add logic myself to check for this in the code, using rem for example (which might be the better solution actually) but I was just wondering if there is a run-time switch for this.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sun, 12 Aug 2012 07:45:52 +0200
Subject: Re: can Ada give run-time error or warning for integer division with non-zero remainder?
Newsgroups: comp.lang.ada

Fortunately not!

Dividing integers uses the integer division. If you want something else, don't use integer division.

Having the outcome of an operation depend on a switch would imply that you cannot predict the result of a program by reading it!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 12 Aug 2012 08:45:03 +0200
Subject: Re: can Ada give run-time error or warning for integer division with non-zero remainder?
Newsgroups: comp.lang.ada

It is not integer division then. You could use integer interval division instead: [3, 3] / [2, 2] = [1, 2].

From: Per Sandberg <per.sandberg@sandat.dyndns.org>
Date: Sun, 12 Aug 2012 09:16:58 +0200
Subject: Re: can Ada give run-time error or warning for integer division with non-zero remainder?
Newsgroups: comp.lang.ada

If you want an new behaviour for "/" on an integer type you could always make a new integer type with the required properties:

```
package Integers is
   type Integer is new Standard.Integer;
   overriding function "/" (L : Integer; R  :
            Integer) return Integer;
   function "/" (L : Standard.Integer; R  :
            Integer) return Integer;
   function "/" (L : Integer; R  :
            Standard.Integer) return Integer;
end Integers;

package body Integers is
   overriding function "/" (L : Integer; R  :
            Integer) return Integer is
   begin
      return Ret : Integer do
         Ret := Integer (Standard.Integer
            (Standard.Integer (L) /
            Standard.Integer (R)));
            if Ret- R /= L then
               raise Constraint_Error
               with L'Img & "/" & R'Img &
               " gives reminder.";
            end if;
      end return;
   end "/";
```

```
  function "/" (L : Standard.Integer; R :
          Integer) return Integer is
  begin
    return Ret : Integer do
      Ret := Integer (L) / R;
    end return;
  end "/";
  function "/" (L : Integer; R :
          Standard.Integer) return Integer is
  begin
    return Ret : Integer do
      Ret := L / Integer (R);
    end return;
  end "/";
end Integers;
```

From: Brad Moore <brad.moore@shaw.ca>
Date: Sun, 12 Aug 2012 08:03:35 -0600
Subject: Re: can Ada give run-time error or warning for integer division with non-zero remainder?
Newsgroups: comp.lang.ada

Or you could simplify this further using some new Ada 2012 features, and eliminate the body.

```
package Integers is
   type Integer is new Standard.Integer;
   overriding function "/" (L : Integer;
      R  : Integer) return Integer is
    (Integer (Standard.Integer (L) /
              Standard.Integer (R)))
     with Post => "/"'Result- R = L;
   function "/" (L : Standard.Integer;
      R  : Integer) return Integer is
     (Integer (L / Standard.Integer (R)))
      with Post => "/"'Result- R = Integer (L);
   function "/" (L : Integer;
      R  : Standard.Integer) return Integer is
     (Integer (Standard.Integer (L) / R))
      with Post => "/"'Result- Integer (R) = L;
end Integers;
```

Using .rlu files in SPARK

From: Ben Hocking <benjaminhocking@gmail.com>
Date: Mon, 13 Aug 2012 07:00:59 -0700
Subject: Using .rlu files in SPARK Ada
Newsgroups: comp.lang.ada

I've been unable to satisfy the following check statement:

```
a2 := x3 - x1;
b := y3 - y1;
--# check b < a2 -> (b rem a2 = b);
```

It generates the following VC:

y3 - y1 < x3 - x1 -> y3 - y1 - (y3 - y1) div (x3 - x1)- (x3 - x1) = y3 - y1

In my .rlu file (which I know is being read because other rules are being applied, and I see it mentioned in my .slg file), I have:

small_rem: y3 - y1 - (y3 - y1) div (x3 - x1)- (x3 - x1) = (y3 - y1) may_be_deduced_from [y3 - y1 < x3 - x1].

(Note that I originally had: small_rem: x rem y = x may_be_deduced_from [x < y]. However, in an effort to rule out any possible source of confusion, I expanded it to exactly what I was seeing in the .siv file.)

Am I doing something incorrectly, or is there some limitation of SPARK here?

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Mon, 13 Aug 2012 16:46:04 +0100
Subject: Re: Using .rlu files in SPARK Ada
Newsgroups: comp.lang.ada

To avoid any additional complications, I'll start by assuming that a2 and b are guaranteed to be Positive (and there should really be something in the rule for that).

For a deduction rule to work, the formula that is deduced has to match the conclusion and the side conditions have to match hypotheses (or be provable from the hypotheses).

This rule would work if the equality was the conclusion and

y3 - y1 < x3 - x1 were a hypothesis.

Since the conclusion is an implication, that is what the rule must be.

So you can either use:

small_rem: y3 - y1 < x3 - x1 -> y3 - y1 - (y3 - y1) div (x3 - x1)- (x3 - x1) = (y3 - y1) may_be_deduced or change the code to:

```
a2 := x3 - x1;
b := y3 - y1;
if b < a2 then
  --# check b rem a2 = b;
  null;
end if;
```

(I have occasionally used null conditional statements to avoid complex implications.)

> (Note that I originally had:
>
> small_rem: x rem y = x may_be_deduced_from [x < y].
>
> However, in an effort to rule out any possible source of confusion, I expanded it to exactly what I was seeing in the .siv file.)
>
> Am I doing something incorrectly, or is there some limitation of SPARK here?

This never had a chance of working because rem is not an FDL operator (which is why the Examiner expands it out.)

However you also need to understand the use of wildcards in rules. This is covered in Section 7 of the Simplifier User Manual, but, briefly, anything in lower case in a rule formula has to match the text of the VC precisely (so the above rule would not work if you changed any of x1, x3, y1, y3 to something else).

Any name in a rule that starts with an upper-case character is a wild-card (aka 'Prolog variable') that can be matched to any expression within a VC, so the ideal rule for your present code is probably: small_rem: A < B -> A - A div B- B = A may_be_deduced .or, to make sure it doesn't get used incorrectly anywhere else:

small_rem: A < B -> A - A div B- B = A may_be_deduced_from[A >= 0, B > 0, goal(checktype(A, integer)), goal(checktype(B, integer))] .

(and there's probably another for negative A and B as well).

Warning: all above expressions typed without any checking with SPARK tools.

An original designer of Ada speaks

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Wed, 15 Aug 2012 22:29:22 +0100
Subject: An original designer of Ada speaks
Newsgroups: comp.lang.ada

… on comp.arch, in the thread

"Re: Have the Itanium critics all been proven wrong?"

Ivan Godard criticizes his work (on Green, I presume) and compares Ada's type system unfavourably with Algol 68.

I have been defending the Lady's honour. 8-)

Networking

From: Shark8 <onewingedshark@gmail.com>
Date: Sun, 12 Aug 2012 13:26:23 -0700
Subject: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

In a related, though only tangential, vein to my other Ada-networking thread, I thought it would be good to ask what [you find] the best way to handle network-communication is.

Is it using Streams? Wrapping up sockets in their own interface-packages and using those? Something I'm not even thinking of? (In short, I'm curious as to how other Ada programmers approach it.)

From: sbelmont700@gmail.com
Date: Sun, 12 Aug 2012 17:10:22 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

Depending on its applicability to your situation, the DSA is perhaps the most elegant solution; simply take the whole thing to the next level higher, and avoid network programming altogether.

From: Shark8 <onewingedshark@gmail.com>
Date: Sun, 12 Aug 2012 17:25:24 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

LOL -- True; I *really- like the underlying ideas for the DSA. (That doesn't mean that I really understand the bits-n-pieces/nitty-gritty… but you have to start somewhere.)

Though there's a lot of programs (or it would be more correct to say 'networks') that aren't DSA but might be useful. Say something like SETI@Home, or EDonkey/Gnutilla/BitTorrent --I know for certain there are Bittorrents being used to distribute people's software, probably most prominently certain Linux flavors-- and they are likely going to need some interfacing support… I really don't see how it would be possible to "overlay the DSA" such that it enables you to treat those as part of your local system.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 13 Aug 2012 09:11:07 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

Depends on the protocol, load and QoS requirements. E.g. in one case we must handle 200K+ connections, in another 80µs is the response time limit.

> "Is it using Streams?"

Rarely. Most network protocols are packet-oriented.

> "Wrapping up sockets in their own interface-packages and using those?"

Certainly. There will be many packages reflecting at least the protocol layers, encapsulating OS-dependent stuff etc.

From: Dmitry A. Kazakov
Date: Mon, 13 Aug 2012 15:11:54 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

> "Does anyone have a favourite approach to TCP/IP?"

TCP vs. UDP changes little (when point to point). You still have packet even when sent over TCP stream. It is difficult to outline a universal solution for all case.

There are a reader and a writer tasks encapsulated into I/O objects. There are packages implementing protocol layers. Each layer object derives from or otherwise uses the lower level object in a mix-in. Lower layers provide operations for the higher levels and define abstract operations or callback to override. Two tasks have the advantage of having it full-duplex (performance, deadlock prevention upon protocol errors), being able to use blocking calls. It may impose some difficulties as you would likely have to route actions from the writer to the reader task.

From: Marc C <mc.provisional@gmail.com>
Date: Mon, 13 Aug 2012 06:12:23 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

> "Does anyone have a favourite approach to TCP/IP?"

After years of working with TCP/IP sockets, my favorite approach now is…not to. Not directly anyway.

There are higher-level messaging protocols that handle most, if not all, of the socket management details for you.

Among them are ZeroMQ[1], along with its Ada binding [2] and YAMI4 [3].

I collected ZeroMQ, AMQP, and STOMP together into a set of text-oriented messaging interfaces called TOMI_4_Ada [4].

While you still have to be cognizant of things like host names and port numbers, by and large that's about the extent of what you need to be aware of when using these high-level protocols.

If I never have to write another setsockopt(), c_select(), and accept() again, I'll be very happy :-)

[1] http://www.zeromq.org
[2] http://www.zeromq.org/bindings:ada
[3] http://www.inspirel.com/yami4
[4] http://sourceforge.net/projects/tomi4ada/

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 13 Aug 2012 15:31:31 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

Hmm, there is not that many things you needed to manage sockets. Setting or clearing TCP_NO_DELAY is not a huge problem, or?

[ZeroMQ —sparre]

Does this really implement protocols, e.g. DLMS, ModBus etc? I didn't read the documentation, but it looks rather like some text messaging or middleware stuff on top of some transport like TCP, than a protocol implementation generator tool (if that were possible).

> "If I never have to write another setsockopt(), c_select(), and accept() again, I'll be very happy :-)"

If Ada provided higher level socket library with an integrated support of protected objects and tasks…

From: Dmitry A. Kazakov
Date: Mon, 13 Aug 2012 18:19:19 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

> "What do you mean?"

The most difficult and unavoidable part about socket programming is tasking. We need reader and writer tasks or else socket-select-driven co-routines. The reader and writer parts have to communicate each other in some intricate way because it is two tasks but one protocol state machine, or one task and many state machines in the case of socket select. There is the issue of blocking socket I/O non-abortable by Ada means. All this incredibly complicates design.

If Ada ever have to support sockets it should be a high level Ada tasking-friendly abstraction, which I am not ready to outline.

From: Patrick <patrick@spellingbeewinnars.org>
Date: Mon, 13 Aug 2012 09:30:18 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

Could you recommend a software project that gets socket programming done right?

From: Marc C <mc.provisional@gmail.com>
Date: Mon, 13 Aug 2012 10:54:35 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

While I'd extract the socket code from an earlier project for reuse, it always needed to be modified a bit to get integrated into the new one. It just gets tedious after awhile. So adopting a more abstract, pre-existing transport protocol just eliminated the need to deal with that.

> "Does this really implement protocols, e.g. DLMS, ModBus etc?"

Nope, ZeroMQ is essentially just a transport layer, which is all the functionality I wanted of it, so that I could stop having to mess around with sockets.

TOMI_4_Ada essentially arose out of my desire to have a simple, consistent Ada library for text-oriented message interfaces. My projects rarely have any need for extensive or high performance protocols, so a nice clean set of client/server and topic-supportive publish/subscribe services meets my needs.

From: Dmitry A. Kazakov
Date: Mon, 13 Aug 2012 20:18:58 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

[Scientific instruments. RS232 / GPIB / TCP/IP. —sparre]

We are using a middleware for that. It abstracts industrial devices and their protocols away, so that the application(s) would not care where the process variables read and written come from. But it is a commercial product.

And of course RS232, GPIB, TCP/IP is as much telling as ISA, PCI, PCIe. The proper description should include the application level and everything down to

the transport, e.g. Steale autopilot, AK (list of commands, their semantics), over TCP/IP.

If you want to communicate industrial devices, TCP/IP would be your least problem. You need a middleware to decouple device-specific stuff from the application logic.

> "Could you recommend a software project that gets socket programming done right?"

I guess there exists some Open Source examples, e.g. AWS (though I did see its sources.

From: Dmitry A. Kazakov
Date: Mon, 13 Aug 2012 20:28:22 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

> "Nope, ZeroMQ is essentially just a transport layer,"

I see. Then it is probably not what the OP wanted.

BTW, did you evaluate the distributed systems Annex vs. this stuff? I am interested in an opinion, because we are using other technology, so that I had no opportunity to evaluate the Annex. Was it too heavy-weight for you?

From: Marc C <mc.provisional@gmail.com>
Date: Mon, 13 Aug 2012 11:59:42 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

Essentially the DSA was ruled out because the systems with which I would be communicating are very unlikely to be done in Ada. And, while I could supply an Ada-implemented "client" with Languages-Other-Than-Ada bindings to it, by offering a "native" client that sends, say JSON across ZeroMQ, it removes a source of hesitancy. In fact, just saying "here's a ZeroMQ port, send me a JSON object with these fields (or an XML doc passing this schema)" and let them go off and build it themselves makes the developers happy. They don't have to depend, then, on *my- API implementation working correctly.

In addition, by employing a standard-ish transport medium--ZeroMQ, AMQP for now--it's much easier to tie in other systems. So long as my system is network-accessible, anyone can get to it--they don't need any client code from me. This therefore enhances interoperability since I supply the only platform-dependent piece, and whoever talks to me can be running whatever OS/HW platform they want.

From: tmoran@acm.org
Date: Mon, 13 Aug 2012 19:11:30 +0000
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

The Claw.Sockets package has a simple Socket_Type with

```
procedure Open(
  Socket: in out Socket_Type;
    -- or Server_Type
  Domain_Name: in String;
     -- or Network_Address_Type
  Port: in Port_Type;
  Timeout: in Duration := 30.0);
```

and Text_IO style Get/Put for Strings, Input/Output for streams, etc.

It also has a non-blocking Async_Socket_Type with overridable When_Connect, When_Readable, etc routines, but I find using blocking sockets with tasks whose structure encodes the state machine is much simpler. All the blocking socket routines allow a timeout parameter, so they in effect poll for data - or a Please_Quit request. (The non-blocking routines are of course also polling, via the Windows message loop.) There are also ftp, pop3, smtp, http, etc packages built on top.

From: Dmitry A. Kazakov
Date: Mon, 13 Aug 2012 22:03:16 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

[Claw.Sockets —sparre]

In an Ada friendly way they should be entry calls, with the semantics of canceling the request when a timed entry call gets timed out. That is when the socket is blocking:

```
select
   Socket.Receive (Packet);
or delay 10.0;  -- Failed to read
or terminate; -- Yes I know, it is illegal to
   --  have terminate + delay
end select;
```

Or maybe other way round, they should be "entry points" to which socket I/O could call to.

For non-blocking I/O there should be a way to have a pseudo-task ran by the events on the socket, rather than scheduled.

From: tmoran@acm.org
Date: Mon, 13 Aug 2012 20:28:58 +0000
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

"Or maybe other way round, they should be "entry points" to which socket I/O could call to."

```
Get(Socket, Timeout => 10.0,
    Item => Buffer, Last => Last);
```

where Last = Buffer'first-1 on timeout, seems simpler to me.

"For non-blocking I/O there should be a way to have a pseudo-task ran by the events on the socket, rather than scheduled."

That's what overidable procedures When_Connect, When_Readable, When_Disconnect, etc are for. They are called when the event occurs.

From: Maciej Sobczak
Date: Tue, 14 Aug 2012 01:39:59 -0700
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

[…]

Or managing multiple transmissions to several targets concurrently? Or having it done in background? Or dealing with serialization across different hardware platforms? Or having the possibility to intermix high-priority traffic with low-priority one? Or having it all integrated with the language so that you can use standard tasking features to wait, time out, etc. on your communication activities? Or raising the conceptual level to data-centric communication where data providers only care about providing data instead of messing with irrelevant low-level stuff? Or…

There's a lot more to networking than sockets and this is where higher-level solutions can be very helpful.

"If Ada provided higher level socket library with an integrated support of protected objects and tasks…"

YAMI4 does provide this level of language integration.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 14 Aug 2012 12:14:01 +0200
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

[When_Connect, When_Readable, When_Disconnect, etc., —sparre]

No, that is upside-down. The point is to write I/O code as if it were synchronous and half-duplex:

```
send
receive
process
send
…
```

Compare it with Ada task. You write a task as if it didn't share the processor, you don't hook at timer interrupts.

From: tmoran@acm.org
Date: Tue, 14 Aug 2012 17:57:24 +0000 (UTC)
Subject: Re: Ada Networking (General/Design)
Newsgroups: comp.lang.ada

Sorry, I misunderstood "pseudo-task ran by the events on the socket" to mean roughly "interrupt-driven". Claw.Sockets has "Async_Socket_Type and procedures When_xxx for that. I agree that it's usually easier to "write I/O code as if it were synchronous and half-duplex:" with the simple Socket_Type and Put, Get, Process, Put...

Conference Calendar

Dirk Craeynest
KU Leuven. Email: Dirk.Craeynest@cs.kuleuven.be

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

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

2012

October 01-04 — 14th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2012), Toronto, Canada. Topics include: Fault-Tolerance and Dependable Systems, Safety and Security, Formal Methods, etc.

Oct 01-05 — **Absolute Software** - Public Ada Course, Carlsbad, CA, USA.

October 01-05 — 10th **International Conference on Software Engineering and Formal Methods** (SEFM'2012), Thessaloniki, Greece. Topics include: programming languages, program analysis and type theory; formal methods for real-time, hybrid and embedded systems; formal methods for safety-critical, fault-tolerant and secure systems; light-weight and scalable formal methods; tool integration; applications of formal methods, industrial case studies and technology transfer; education and formal methods; etc.

October 08-11 — 31st IEEE **International Symposium on Reliable Distributed Systems** (SRDS'2012), Irvine, California, USA. Topics include: distributed systems design, development and evaluation, with emphasis on reliability, availability, safety, security, trust and real time; high-confidence and safety-critical systems; distributed objects and middleware systems; formal methods and foundations for dependable distributed computing; evaluations of dependable distributed systems; etc.

October 12-13 — 35th IEEE **Software Engineering Workshop** (SEW-35), Heraclion, Crete, Greece. Topics include: metrics and experience reports; software quality assurance; formal methods and formal approaches to software development; software engineering processes and process improvement; real-time software engineering; software maintenance, reuse, and legacy systems; etc.

October 15-18 — 19th **Working Conference on Reverse Engineering** (WCRE'2012), Kingston, Ontario, Canada. Topics include: theory and practice of recovering information from existing software and systems, such as program comprehension, mining software repositories, empirical studies in reverse engineering, redocumenting legacy systems, reverse engineering tool support, reengineering to distributed architectures, software architecture recovery, program analysis and slicing, reengineering patterns, program transformation and refactoring, etc.

☺ October 19-26 — ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2012), Tucson, Arizona, USA. Topics include: the intersection of programming, programming languages, and software engineering; areas such as programming methods, design and analysis, testing, concurrency, program analysis, empirical studies, and new programming languages; all aspects of software construction and delivery, all factions of programming technologies.

- ☺ October 21 — 4th **Workshop on Evaluation and Usability of Programming Languages and Tools** (PLATEAU'2012). Topics include: methods, metrics and techniques for evaluating the usability of languages and language tools, such as empirical studies of programming languages, methodologies and philosophies behind language and tool evaluation, software design metrics and their relations to the underlying language, user studies of language features and software engineering tools, critical comparisons of programming paradigms, tools to support evaluating programming languages, etc.
- October 21 — **Workshop on Relaxing Synchronization for Multicore and Manycore Scalability** (RACES'2012).

October 25-27 14th IEEE **International High Assurance Systems Engineering Symposium** (HASE'2012), Omaha, Nebraska, USA. Topics include: integrated approaches for assuring reliability, availability, integrity, privacy, confidentiality, safety, and real-time performance of complex systems; and methods for assessing assurance levels of these systems to a high degree of confidence.

October 29-31 12th **Annual International Conference on New Technologies of Distributed Systems** (NOTERE'2012), Anglet, France. Topics include: distributed systems and middleware, particularly those based on objects, components, agents, including real time and embedded; modeling Languages or DSL dedicated to distributed systems; modeling, formal and semi-formal methods, and tools for distributed systems; reliability and scalability of distributed systems; etc.

November 10-16 25th **International Conference for High Performance Computing, Networking, Storage and Analysis** (SC'2012), Salt Lake City, Utah, USA. Topics include: Applications, Languages, and Programming Environments; Innovation in HPC and Emerging Technologies; etc.

November 10-17 20th ACM SIGSOFT **International Symposium on the Foundations of Software Engineering** (FSE'2012), Research Triangle Park, North Carolina, USA. Topics include: Architecture and design; Components, services, and middleware; Distributed, parallel and concurrent software; Embedded and real-time software; Empirical studies of software engineering; Formal methods; Reverse engineering and maintenance; Security, safety and reliability; Software tools and development environments; Specification and verification; etc.

November 12-16 14th **International Conference on Formal Engineering Methods** (ICFEM'2012), Kyoto, Japan. Topics include: abstraction and refinement; software verification; program analysis; formal methods for robotics, cyber-physical systems, medical devices, aeronautics, railway; formal methods for software safety, security, reliability and dependability; experiments involving verified systems; formal model-based development and code generation; etc.

November 12 1st **International Workshop on Formal Techniques for Safety-Critical Systems** (FTSCS'2012). Topics include: case studies and experience reports on the use of formal methods for analyzing safety-critical systems, including avionics, automotive, medical, and other kinds of safety-critical and QoS-critical systems; methods, techniques and tools to support automated analysis, certification, debugging, etc., of complex safety/QoS-critical systems; etc.

☺ November 14-15 **Automotive - Safety & Security 2012**, Karlsruhe, Germany. Organized by Gesellschaft für Informatik mit den Fachgruppen Ada, etc, and Ada-Deutschland. Topics include (in German): Zuverlässigkeit und Sicherheit für betriebskritische Software und IT-Systeme; Evaluation u. Qualifikation von Sicherheitseigenschaften automobiler Plattform- und Applikationssoftware; Werkzeuge zur Verbesserung der Zuverlässigkeit im Software Life Cycle; Multi-Core-Architekturen; Fortschritte bei Normen und Standardisierungen; etc.

☺ November 16 SC2012 - 5th **International Workshop on Multi-Core Computing Systems** (MuCoCoS'2012), Salt Lake City, Utah. Theme: "Performance Portability and Tuning". Topics include: portable programming models, languages and compilation techniques; case studies highlighting performance portability and tuning; etc. Deadline for registration: October 17, 2012.

November 18-23 7th **International Conference on Software Engineering Advances** (ICSEA'2012), Lisbon, Portugal. Topics include: Advances in fundamentals for software development; Advanced mechanisms for software development; Advanced design tools for developing software; Software security, privacy, safeness; Specialized software advanced applications; Open source software; Agile software techniques; Software deployment and maintenance; Software engineering techniques, metrics, and formalisms; Software economics, adoption, and education; etc

November 27-30 23rd IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2012), Dallas, Texas, USA. Topics include: reliability, availability, and safety of software systems; validation, verification, testing and dynamic analysis; software quality and productivity; software security; dependability, survivability, and resilience of software systems; open source software reliability engineering; supporting tools and automation; industry best practices; empirical studies; etc.

November 29-30 **Many-core Applications Research Community Symposium** (MARC'2012), Aachen, Germany. Topics include: dealing with legacy software on novel many-core architectures; experiences porting,

running, or developing applications; traditional and new programming models for novel many-core hardware; etc.

♦ Dec 02-06 ACM SIGAda Annual **International Conference on High Integrity Language Technology** (HILT'2012), Boston, Massachusetts, USA.

December 04-07 19th **Asia-Pacific Software Engineering Conference** (APSEC'2012), Hong Kong, China. Theme: "Software Engineering for the Evolving World". Topics include: Software architecture and design, SE methodologies, Software analysis and understanding, Software verification and validation, Software maintenance and evolution, Software quality, Software process and standards, Software security and reliability, SE environments and tools, SE education, Distributed and parallel software systems, Embedded and real-time software systems, Component based SE, Product-line SE, Formal methods in SE, Emerging SE methods, etc.

☺ December 05-07 33th IEEE **Real-Time Systems Symposium** (RTSS'2012), San Juan, Porto Rico. Topics include: all aspects of real-time systems design, analysis, implementation, evaluation, and experiences.

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

December 13-15 10th **Asian Symposium on Programming Languages and Systems** (APLAS'2012), Kyoto, Japan. Topics include: both foundational and practical issues in programming languages and systems, such as semantics, design of languages, type systems, compilers, program analysis, verification, software security, concurrency, parallelism, tools for programming, verification, and implementation, etc.

☺ December 14-16 13th **International Conference on Parallel and Distributed Computing, Applications, and Techniques** (PDCAT'2012), Beijing, China. Topics include: all areas of parallel and distributed computing; Reliability, and fault-tolerance; Formal methods and programming languages; Software tools and environments; Parallelizing compilers; Component-based and OO Technology; Parallel/distributed algorithms; Task mapping and job scheduling; etc.

☺ December 17-19 18th IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2012), Singapore. Topics include: Parallel and Distributed Applications and Algorithms; Multi-core and Multithreaded Architectures; Security and Privacy; Dependable and Trustworthy Computing and Systems; Real-Time Systems; Embedded systems; etc.

December 18-21 19th IEEE **International Conference on High Performance Computing** (HiPC'2012), Pune, India. Topics include: Parallel and Distributed Algorithms/Systems, Parallel Languages and Programming Environments, Hybrid Parallel Programming with GPUs and Accelerators, Scheduling, Fault-Tolerant Algorithms and Systems, Scientific/Engineering/Commercial Applications, Compiler Technologies for High-Performance Computing, Software Support, etc. Deadline for early registration: November 14, 2012.

2013

☺ January 23-25 40th ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2013), Rome, Italy. Topics include: fundamental principles and important innovations in the design, definition, analysis, transformation, implementation and verification of programming languages, programming systems, and programming abstractions.

Jan 20-21 ACM SIGPLAN **Workshop on Partial Evaluation and Program Manipulation** (PEPM'2013). Topics include: Program and model manipulation techniques (such as: partial evaluation, slicing, symbolic execution, refactoring, ...); Program analysis techniques that are used to drive program/model manipulation (such as: abstract interpretation, termination checking, type systems, ...); Techniques that treat programs/models as data objects (including: metaprogramming, generative programming, model-driven program generation and transformation, ...); etc. Application of the above techniques including case studies of program manipulation in real-world (industrial, open-source) projects and software development processes, descriptions of robust tools capable of effectively handling realistic applications, benchmarking.

Jan 29 - Feb 01 11th **Australasian Symposium on Parallel and Distributed Computing** (AusPDC'2013), Adelaide, South Australia. Topics include: multicore systems; GPUs and other forms of special purpose

processors; middleware and tools; parallel programming models, languages and compilers; runtime systems; reliability, security, privacy and dependability; applications; etc.

January 21-23 6th **India Software Engineering Conference** (ISEC'2013), New Delhi, India. Topics include: static analysis, specification and verification, model driven software engineering, software architecture and design, tools and environments, maintenance and evolution, component based software engineering, object-oriented technology, distributed software development, software engineering education, software security, mining software repositories, embedded and real-time systems, etc.

☺ January 23-27 18th ACM SIGPLAN **Symposium on Principles and Practice of Parallel Programming** (PPoPP'2013), Shenzhen, China. Topics include: formal analysis and verification; parallel programming languages; compilers and runtime systems; development, analysis, or management tools; concurrent data structures; synchronization and concurrency control; software engineering for parallel programs; software issues for multicore and multithreaded processors; task mapping and scheduling; etc.

Feb 27 – Mar 01 5th **International Symposium on Engineering Secure Software and Systems** (ESSoS'2013), Paris, France. Topics include: security architecture and design for software and systems; specification formalisms for security artifacts; verification techniques for security properties; systematic support for security best practices; programming paradigms for security; processes for the development of secure software and systems; support for assurance, certification and accreditation; etc.

Feb 27 – Mar 01 21st Euromicro **International Conference on Parallel, Distributed and Network-Based Computing** (PDP'2013), Belfast, Northern Ireland, UK. Topics include: embedded parallel and distributed systems, multi- and many-core systems, programming languages and environments, runtime support systems, dependability and survivability, advanced algorithms and applications, etc.

March 06-09 44th ACM **Technical Symposium on Computer Science Education** (SIGCSE'2013), Denver, Colorado, USA.

March 18-22 28th ACM **Symposium on Applied Computing** (SAC'2013), Coimbra, Portugal.

- ☺ Mar 18-22 **Track on Programming Languages** (PL'2013). Topics include: Compiling Techniques, Formal Semantics and Syntax, Garbage Collection, Language Design and Implementation, Languages for Modeling, Model-Driven Development New Programming Language Ideas and Concepts, Practical Experiences with Programming Languages, Program Analysis and Verification, Programming Languages from All Paradigms, etc.
- ☺ Mar 18-22 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2013). Topics include: Aspects and components; Distribution and concurrency; Formal verification; Integration with other paradigms; Interoperability, versioning and software evolution and adaptation; Language design and implementation; Modular and generic programming; Static analysis; Type systems; etc. Deadline for paper submissions: October 31, 2012 (research abstracts).
- March 18-22 **Track on Software Verification and Testing** (SVT'2013). Topics include: tools and techniques for verification of large scale software systems, real world applications and case studies applying software verification, static and run-time analysis, refinement and correct by construction development, software certification and proof carrying code, etc.

March 18-22 6th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2013), Luxembourg. Topics include: domain specific testing, security testing, embedded-software testing, testing concurrent software, testing large-scale distributed systems, testing in multi-core environments, quality assurance, empirical studies, agile/iterative/incremental testing processes, testing of open source and third-party software, software reliability, formal verification, experience reports, etc. Deadline for submissions: November 14, 2012 (Ph.D. symposium), February 9, 2013 (posters).

March 25-29 12th **International Conference on Aspect-Oriented Software Development** (AOSD'2013), Fukuoka, Japan. Topics include: Complex systems; Software design and engineering (evolution, economics, composition, methodology, ...); Programming languages (language design, compilation and interpretation, verification and static program analysis, formal languages, execution environments and dynamic weaving, ...); Varieties of modularity (model-driven development, generative programming, software product lines, contracts and components, ...); Tools (evolution and reverse engineering,

crosscutting views, refactoring, ...); Applications (distributed and concurrent systems, middleware, runtime verification, ...); etc.

April 01-05 6th **Latin-American Symposium on Dependable Computing** (LADC'2013), Rio de Janeiro, Brazil. Deadline for submissions: January 18, 2013 (fast abstracts, student forum and industrial track).

♦ April 17-19 16th **International Real-Time Ada Workshop** (IRTAW'2013), York, England. Deadline for position paper submissions: February 1, 2013.

☺ May 18-26 35th **International Conference on Software Engineering** (ICSE'2013), San Francisco, USA. Theme: "Software Engineering Ideas to Change the World". Deadline for submissions: November 2, 2012 (workshop proposals tutorial proposals software engineering in practice papers software engineering education papers new ideas and emerging results papers doctoral symposium submissions formal demonstrations), December 17, 2012 (ACM Student Competition), January 30, 2013 (SCORE full project submission).

☺ May 20-24 27th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2013), Boston-Cambridge, USA. Topics include: all areas of parallel and distributed processing, such as parallel and distributed algorithms, applications of parallel and distributed computing, parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, parallel programming paradigms, and programming environments and tools, etc.

♦ June 10-14 18th **International Conference on Reliable Software Technologies – Ada-Europe'2013**, Berlin, Germany. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN (requests pending). Deadline for submissions: December 3, 2012 (papers, tutorials, workshops), January 14, 2013 (industrial presentations).

July 01-03 18th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2013), Canterbury, Kent, UK.

☺ September 10-13 **International Conference on Parallel Computing 2013** (ParCo'2013), München, Germany. Topics include: all aspects of parallel computing, including applications, hardware and software technologies as well as languages and development environments, in particular Parallel programming languages, compilers, and environments; Tools and techniques for generating reliable and efficient parallel code; Best practices of parallel computing on multicore, manycore, and stream processors; etc. Deadline for submissions: February 28, 2013 (extended abstracts), March 31, 2013 (mini-symposia).

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

Association for Computing Machinery

Advancing Computing as a Science & Profession

HILT 2012: HIGH INTEGRITY LANGUAGE TECHNOLOGY

ACM SIGAda's Annual International Conference

December 2–6, 2012 / Boston, Massachusetts / Advance Program

High integrity software must not only meet correctness and performance criteria but also satisfy stringent safety and/or security demands, typically entailing certification against a relevant standard.

A significant factor affecting whether and how such requirements are met is the chosen language technology and its supporting tools: not just the programming language(s) but also languages for expressing specifications, program properties, domain models, and other attributes of the software or overall system.

HILT 2012 provides a forum for the leading experts from academia/research, industry, and government to present their latest findings in designing, implementing, and using language technology for high integrity software.

Sponsored by SIGAda, ACM's Special Interest Group on the Ada Programming Language, in cooperation with SIGCSE, SIGPLAN, SIGSOFT, SIGBED, Ada-Europe, and the Ada Resource Association.

KEYNOTE TOPICS / FEATURED SPEAKERS

High-Assurance Cyber Military Systems (HACMS): High-Assurance Vehicles

KATHLEEN FISHER
DARPA Information Innovation Office

Challenges for Safety-Critical Software

NANCY LEVESON
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
Engineering Systems Division

Programming the Turing Machine

BARBARA LISKOV
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Hardening Legacy C/C++ Code

GREG MORRISETT
Harvard University
School of Engineering and Applied Sciences

Programming Language Life Cycles

GUY L. STEELE, JR.
Oracle Labs

CORPORATE SPONSORS

PLATINUM LEVEL

AdaCore
The GNAT Pro Company

SILVER LEVEL

Ellidiss Software
TNI Europe Limited

LDRA

TASC

PRE-CONFERENCE TUTORIALS / December 2–3

SUNDAY
Pre-Conference Tutorials

SF1— Full Day / 9:00 AM–5:30 PM
Bo I. Sandén /
Colorado Technical University
Design of Multitask Software: The Entity-Life Modeling Approach

SA1—Morning / 9:00 AM–12:30 PM
Jason Belt, Patrice Chalin, John Hatcliff, and Robby / Kansas State University
Leading-Edge Ada Verification Technologies: Highly Automated Ada Contract Checking Using Bakar Kiasan

SA2—Morning / 9:00 AM–12:30 PM
Ed Colbert / Absolute Software
Ada 2012 Contracts and Aspects

SP1—Afternoon / 2:00 PM–5:30 PM
Johannes Känig / AdaCore
Leading-Edge Ada Verification Technologies: Combining Testing and Verification with GNATTest and GNATProve—The Hi-Lite Project

SP2—Afternoon / 2:00 PM–5:30 PM
Ed Colbert / Absolute Software
Object-Oriented Programming with Ada 2005 and 2012

MONDAY
Pre-Conference Tutorials

MF1—Full Day / 9:00 AM–5:30 PM
Nancy Leveson, Cody Fleming, and John Thomas /
Massachusetts Institute of Technology
Safety of Embedded Software

MA1—Morning / 9:00 AM–12:30 PM
K. Rustan M. Leino / Microsoft Research
Developing Verified Programs with Dafny

MA2—Morning / 9:00 AM–12:30 PM
Ricky E. Sward / The MITRE Corporation
Jeff Boleng /
Software Engineering Institute
Service-Oriented Architecture (SOA) Concepts and Implementations

MP1—Afternoon / 2:00 PM–5:30 PM
Tucker Taft / AdaCore
Multicore Programming using Divide-and-Conquer and Work Stealing

MP2—Afternoon / 2:00 PM–5:30 PM
Kevin Nilsen / Atego
Understanding Dynamic Memory Management in Safety Critical Java

TECHNICAL PROGRAM / December 4–6

TUESDAY
Analyzing and Proving Programs

9:00 AM–10:30 AM
Greetings
SIGAda and Conference Officers
Keynote Address
Barbara Liskov,
Massachusetts Institute of Technology
Programming the Turing Machine

10:30 AM–11:00 AM Break / Exhibits

11:00 AM–12:30 PM
Program Verification at Compile-Time

K. Rustan M. Leino
Program Proving Using Intermediate Verification Languages (IVLs) like Boogie and Why3

C. Dross, J. Känig, and E. Schonberg
Hi-Lite: The Convergence of Compiler Technology and Program Verification

Industrial/Sponsor Presentation

12:30 PM–2:00 PM Break / Exhibits

2:00 PM–3:30 PM
Keynote Address
Greg Morrisett, Harvard University
Hardening Legacy C/C++ Code

3:30 PM–4:00 PM Break / Exhibits

4:00 PM–5:30 PM
Advancing Compilation Technology

V. Pucci and E. Schonberg
The Implementation of Compile-Time Dimensionality Checking

H. Kirtchev
A Robust Implementation of Ada's Finalizable Controlled Types

5:30 PM–7:00 PM Break

7:00 PM–10:00 PM
Social Event / Dinner

WEDNESDAY
Security and Safety

9:00 AM–10:30 AM
Announcements
SIGAda Awards
Ricky E. Sward, SIGAda Chair
Keynote Address
Kathleen Fisher, DARPA
HACMS: High-Assurance Vehicles

10:30 AM–11:00 AM Break / Exhibits

11:00 AM–12:30 PM
Languages and Security

M. Norrish
Formal Verification of the seL4 Microkernel

D. S. Hardin
DSL for Cross-Domain Security

Industrial/Sponsor Presentation

12:30 PM–2:00 PM Break / Exhibits

2:00 PM–3:30 PM
Keynote Address
Nancy Leveson,
Massachusetts Institute of Technology
Challenges for Safety-Critical Software

3:30 PM–4:00 PM Break

4:00 PM–5:30 PM
Languages and Safety

TRACK 1
Industrial Session on Safety

K. Nilsen
Real-Time Java in the Modernization of the Aegis Weapon System

J. O'Leary
Software for FAA's Automatic Data Comm Between Air Traffic Controller and Pilot

Industrial/Sponsor Presentation

Industrial/Sponsor Presentation

TRACK 2
Real Time Systems

G. Bosch
Synchronization Cannot Be a Library

S. Li et al.
Applicability of RT Schedulability Analysis on a Software Radio Protocol

Industrial/Sponsor Presentation

5:30 PM–7:00 PM Break

7:00 PM–10:00 PM
Workshops /
Birds-of-a-Feather Sessions

THURSDAY
Designing and Implementing Languages

9:00 AM–10:30 AM
Announcements
Best Paper and Student Paper Awards
Jeff Boleng, HILT 2012 Program Co-Chair
Keynote Address
Guy L. Steele, Jr., Oracle Labs
Programming Language Life Cycles

10:30 AM–11:00 AM Break

11:00 AM–1:00 PM
Compiler Certification Issues

D. Eilers and T. Koskinen
Adapting ACATS for Use with Run-Time Checks Suppressed

Panel on Compiler Certification
L. Berringer (CompCert), R. Brukardt (Ada), T. Plum (C, C++, Java)

Announcements
(Ada-Europe 2013, SIGAda 2013)

Closing Remarks and
Conference Adjournment

To register online, and for more information and updates, visit

www.sigada.org/conf/hilt2012

16TH INTERNATIONAL REAL-TIME ADA WORKSHOP (IRTAW 2013)

17-19 April 2013 – Kings Manor, York, England

http://www.cs.york.ac.uk/~andy/IRTAW2013/

CALL FOR PAPERS

Since the late Eighties the **International Real-Time Ada Workshop** series has provided a forum for identifying issues with real-time system support in Ada and for exploring possible approaches and solutions, and has attracted participation from key members of the research, user, and implementer communities worldwide. Recent IRTAW meetings have significantly contributed to the Ada 2005 and Ada 2012 standards, especially with respect to the tasking features, the real-time and high-integrity systems annexes, and the standardization of the Ravenscar profile..

In keeping with this tradition, the goals of IRTAW-16 will be to:

- review the current status of the Ada 2012 Issues that are related with the support of real-time systems;
- examine experiences in using Ada for the development of real-time systems and applications, especially – but not exclusively – those using concrete implementation of the new Ada 2012 real-time features;
- report on or illustrate implementation approaches for the real-time features of Ada 2012;
- consider the added value of developing other real-time Ada profiles in addition to the Ravenscar profile;
- examine the implications to Ada of the growing use of multiprocessors in the development of real-time systems, particularly with regard to predictability, robustness, and other extra-functional concerns;
- examine and develop paradigms for using Ada for real-time distributed systems, with special emphasis on robustness as well as hard, flexible and application-defined scheduling;
- consider the definition of specific patterns and libraries for real-time systems development in Ada;
- identify how Ada relates to the certification of safety-critical and/or security-critical real-time systems;
- examine the status of the Real-Time Specification for Java and other languages for real-time systems development, and consider user experience with current implementations and with issues of interoperability with Ada in embedded real-time systems;
- consider the lessons learned from industrial experience with Ada and the Ravenscar Profile in actual real-time projects;
- consider the language vulnerabilities of the Ravenscar and full language definitions..

Participation at **IRTAW-16** is by invitation following the submission of a position paper addressing one or more of the above topics or related real-time Ada issues. Alternatively, anyone wishing to receive an invitation, but for one reason or another is unable to produce a position paper, may send in a one-page position statement indicating their interests. Priority will, however, be given to those submitting papers.

Submission requirements

Position papers should not exceed ten pages in typical IEEE conference layout, excluding code inserts. All accepted papers will appear, in their final form, in the Workshop Proceedings, which will be published as a special issue of Ada Letters (ACM Press). Selected papers will also appear in the Ada User Journal.

Please submit position papers, in PDF format, to the Program Chair by e-mail: alan.burns@york.ac.uk

Important Dates

Receipt of Position Paper: 1 February 2013	Final Copy of Paper: 1 April 2013
Notification of Acceptance: 1 March 2013	Workshop Date: 17-19 April 2013

Call for Papers

18th International Conference on Reliable Software Technologies

Ada-Europe 2013

10-14 June 2013, Berlin, Germany

http://www.ada-europe.org/conference2013

Special Interest Group Ada of theGerman Informatics Society

Conference and Program Co-Chairs

Hubert B. Keller
Karlsruhe Institute of Technology
hubert.keller@kit.edu

Erhard Plödereder
University of Stuttgart
ploedere@iste.uni-stuttgart.de

Tutorial Chair

Jürgen Mottok
Regensburg University of Applied Sciences
Juergen.Mottok@hs-regensburg.de

Industrial Chair

Jørgen Bundgaard
Ada in Denmark
jb@ada-dk.org

Exhibition Chair

Peter Dencker
ETAS GmbH
peter.dencker@etas.com

Publicity Chair

Dirk Craeynest
Ada-Belgium & KU Leuven
Dirk.Craeynest@cs.kuleuven.be

Local Chair

Raúl Rojas
FU Berlin
Raul.Rojas@fu-berlin.de

Local Organizer

Christine Harms
christine.harms@ccha.de

In cooperation (requests pending) with
ACM SIGAda, SIGBED, SIGPLAN

General Information

The 18th International Conference on Reliable Software Technologies – Ada-Europe 2013 will take place in Berlin, Germany. Following its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical and industrial sessions, along with parallel tutorials and workshops on Monday and Friday.

Schedule

3 December 2012	Submission of regular papers, tutorial and workshop proposals
14 January 2013	Submission of industrial presentation proposals
11 February 2013	Notification of acceptance to all authors
10 March 2013	Camera-ready version of regular papers required
11 May 2013	Industrial presentations, tutorial and workshop material required

Topics

The conference has successfully established itself as an international forum for providers, practitioners and researchers into reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies.

To mark the completion of the **Ada 2012** standard revision process, contributions are sought that discuss experiences with the revised language.

Topics of interest to this edition of the conference include but are not limited to:

- **Multicore Programming**: Reliable Parallel Software, Scheduling on Multi-Core Systems, Compositional Parallelism Models, Performance Modelling, Deterministic Debugging.
- **Real-Time and Embedded Systems**: Real-Time Software, Architecture Modelling, HW/SW Co-Design, Reliability and Performance Analysis.
- **Theory and Practice of High-Integrity Systems**: Distribution, Fault Tolerance, Security, Reliability, Trust and Safety, Languages Vulnerabilities.
- **Software Architectures**: Design Patterns, Frameworks, Architecture-Centered Development, Component and Class Libraries, Component-based Design and Development.
- **Methods and Techniques for Software Development and Maintenance**: Requirements Engineering, Object-Oriented Technologies, Model-driven Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- **Enabling Technologies**: Compilers, Support Tools (Analysis, Code/Document Generation, Profiling), Run-time Systems, Distributed Systems, Ada and other Languagesfor Reliable Systems.
- **Software Quality**: Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Energy, Games and Serious Games, etc.
- **Experience Reports in Reliable System Development**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **Experiences with Ada and its Future**: New Language Features, Implementation and Use Issues; Positioning in the Market and in Education; where should Ada stand in the Software Engineering Curriculum; Lessons Learned on Ada Education and Training Activities with bearing on any of the conference topics.

Program Committee

Ted Baker, US National Science Foundation, USA
Johann Blieberger, Technische Universität Wien, Austria
Bernd Burgstaller, Yonsei University, Korea
Alan Burns, University of York, UK
Rod Chapman, Altran Praxis Ltd, UK
Dirk Craeynest, Ada-Belgium& KU Leuven, Belgium
Juan A. de la Puente, Universidad Politécnica de Madrid, Spain
Franco Gasperoni, AdaCore, France
Michael González Harbour, Universidad de Cantabria, Spain
Xavier Grave, Centre National de la Recherche, France
Christoph Grein, Ada Germany, Germany
J. Javier Gutiérrez, Universidad de Cantabria, Spain
Peter Hermann, Universität Stuttgart, Germany
Jérôme Hugues, ISAE Toulouse, France
Pascal Leroy, Google, Switzerland
Albert Llemosí, Universitat de les IllesBalears, Spain
Kristina Lundqvist, Mälardalen University, Sweden
Franco Mazzanti, ISTI-CNR Pisa, Italy
John McCormick, University of Northern Iowa, USA
Stephen Michell, Maurya Software, Canada
Luís Miguel Pinho, CISTER Research Centre/ISEP, Portugal
Jürgen Mottok, Regensburg University of Applied Sciences, Germany
Manfred Nagl, RWTH Aachen University, Germany
Laurent Pautet, Telecom ParisTech, France
Jorge Real, UniversitatPolitécnica de València, Spain
Jean-Pierre Rosen, Adalog, France
José Ruiz, AdaCore, France
Ed Schonberg, AdaCore, USA
Tucker Taft, AdaCore, USA
Theodor Tempelmeier, Univ. of Applied Sciences Rosenheim, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Tullio Vardanega, Università di Padova, Italy
Juan Zamorano, Universidad Politécnica de Madrid, Spain

Indutrial Committee

JørgenBundgaard, Rambøll Danmark, Denmark
Jacob Sparre Andersen, JSA, Denmark
Jamie Ayre, AdaCore, France
Ian Broster, Rapita Systems, UK
Rod Chapman, Altran Praxis Ltd, UK
Dirk Craeynest, Ada-Belgium& KU Leuven, Belgium
Michael Friess, AdaCore, France
Ismael Lafoz, Airbus Military, Spain
Ahlan Marriott, White-Elephant GmbH, Switzerland
Steen Ulrik Palm, Terma, Denmark
Paolo Panaroni, Intecs, Italy
Paul Parkinson, Wind River, UK
Ana Isabel Rodríguez, GMV, Spain
Jean-Pierre Rosen, Adalog, France
AlokSrivastava, TASC Inc, USA
Claus Stellwag, Elektrobit AG, Germany
Jean-Loup Terraillon, European Space Agency, The Netherlands
Rod White, MBDA, UK

Call for Regular Papers

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall be in English, complete and not exceeding 14 LNCS-style pages in length. Authors should submit their work via the EasyChair conference system (*http://www.easychair.org/conferences/?conf=ae13*). The format for submission is solely PDF. For any remaining questions, please contact a*Program Co-Chair*.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the start of the conference. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by March 10, 2013. For format and style guidelines authors should refer to the following URL: http://www.springer.de/comp/lncs/authors.html. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The conference is ranked class A in the CORE ranking, is among the top quarter of CiteSeerX Venue Impact Factor, and listed in DBLP, SCOPUS and the Web of Science Conference Proceedings Citation index, among others.

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Industrial Presentations

The conference also seeks industrial presentations which deliver value and insight, but may not fit the selection process for regular papers. Authors of industrial presentations are invited to submit an overview (at least 1 full page in length) of the proposedpresentation by January 14, 2013, via the EasyChair conference system (*http://www.easychair.org/conferences/?conf=ae13*).The *Industrial Committee* will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by May 13, 2013, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the *Industrial Chair* directly.

Call for Tutorials

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

Call for Workshops

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

Call for Exhibitors

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

Grant for Reduced Student Fees

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

FOR IMMEDIATE RELEASE

Ada-Europe Announces First "Ada Way" Award Winners and "Try and Beat Me" Challenge

Brussels, Belgium (July 30, 2012) – Ada-Europe, www.ada-europe.org, the international organization that promotes the knowledge and use of the Ada programming language in European academia, research and industry, launched "The Ada Way" annual student programming contest in September 2010. The first challenge was to build a software simulator of a football (soccer) match. The submitted code had to include a software core implementing the logic of the simulation, and read-write graphical panels for interactive team management.

The evaluation committee chose one of the submissions made until April 2012, which at the recent Ada-Europe 2012 conference in Stockholm was proclaimed the reference implementation. The winning student team, formed by *Ricardo Aguirre Reyes, Andrea Graziano, Marco Teoli, and Alberto Zuccato*, received a laminated Ada Way Award donated by Ada-Europe to commend the outstanding quality of their submission.

In evaluating the authors' submission the evaluation committee reported: "*This implementation of the Ada Way Soccer Simulation reveals extraordinary care and engineering skill, and represents a working, scalable, well-documented, and well-structured solution. From reading the technical documentation, it is clear that the development team faced many challenges, and in every case determined an appropriate solution through a combination of thoughtful analysis, experimentation, and clever design."* The story of their implementation will be told in a forthcoming issue of the *Ada User Journal*, the quarterly magazine of Ada-Europe. In due course, the winning team will receive all elements of the prize attached to their fine achievement.

Today, Ada-Europe is pleased to announce that the full source of the reference implementation is posted on the Ada Way page, www.ada-europe.org/AdaWay, along with its accompanying technical specification, user manual and build instructions, a short demo video clip and an image of the award.

The reference implementation is now proposed for a **"Try and Beat Me"** open-ended challenge: any student team willing to take that challenge is invited to make a submission that attempts to improve over the reference implementation under any of the evaluation criteria listed on the Ada Way page. On 15 May of every year, any such new submission will be evaluated and the best one will be awarded a minor prize and will replace the previous reference submission in the continuation of the try-and-beat-me challenge.

The evaluation will be performed by a team of distinguished Ada experts comprised of: John Barnes (author of the famous *Programming in Ada* books), S. Tucker Taft (leader of the *Ada 95* language revision), Pascal Leroy (leader of the *Ada 2005* language revision), Ed Schonberg (co-author of the open-source *GNAT Ada* compiler and toolset), Joyce Tokar (convenor of the *ISO working group* on the Ada language standards), etc.

The winning team will be announced at the Ada-Europe yearly conference subsequent to the cut-off date at which submissions entered the challenge. The prize for this challenge includes a framed award, an Ada book of choice, visibility in electronic and printed media, one free registration and a monetary grant of up to EUR 1000 for the winning team to use for collective participation at any future Ada-Europe conference of choice within two calendar years after selection for the prize.

Ada-Europe wants the competition to be fun and instructive. The implementation does not need to be 100% Ada, but the essence must of course be. Tullio Vardanega, president of Ada-Europe, stated: "*The winning submission must be a reference for good Ada programming, software design, and innovation."*

For all details, please refer to the official web page of "The Ada Way", www.ada-europe.org/AdaWay.

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of the Ada programming language in academia, research and industry in Europe. Its flagship event is the annual international Ada-Europe conference on reliable software technologies, a high-quality technical and scientific event that has been successfully running in the current format for the last 17 years. Ada-Europe has member organizations all over the continent, in Belgium, Denmark, France, Germany, Spain, Sweden, and Switzerland, as well as individual members in many other countries. For more information about Ada-Europe, its charter, activities and sponsors, please visit its web site.

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

Press contact

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

Rationale for Ada 2012: 4 Tasking and Real-Time

John Barnes

John Barnes Informatics, 11 Albert Road, Caversham, Reading RG4 7AN, UK; Tel: +44 118 947 4125; email: jgpb@jbinfo.demon.co.uk

Abstract

This paper describes various improvements in the tasking and real-time areas for Ada 2012.

The most important is perhaps the recognition of the need to provide control over task allocation on multiprocessor architectures.

There are also various improvements to the scheduling mechanisms and control of budgets with regard to interrupts.

An interesting addition to the core language is the ability to specify restrictions on how a procedure of a synchronized interface is to be implemented.

Keywords: rationale, Ada 2012.

1 Overview of changes

The WG9 guidance document [1] identifies real-time systems as an important application area for Ada. In particular it says that attention should be paid to

> improving the capabilities of Ada on multicore and multiprocessor architectures.

Ada 2012 does indeed address the issues of multiprocessors as well as other real-time improvements.

The following Ada Issues cover the relevant changes and are described in detail in this paper:

30 Requeue on synchronized interfaces

117 Memory barriers and Volatile objects

166 Yield for non-preemptive dispatching

167 Affinities for programs on multiprocessor platforms

168 Extended suspension objects

169 Group budgets for multiprocessors

170 Monitoring time spent in interrupt handlers

171 Pragma CPU and Ravenscar profile

174 Implement task barriers in Ada

215 Pragma Implemented should be an aspect

278 Set_CPU called during a protected action

These changes can be grouped as follows.

First there are a number of improvements and additions to the scheduling mechanisms (166, 168, 174). These are in the Real-Time Systems annex (D).

A number of additions recognise the importance of the widespread introduction of multiprocessors and provide mechanisms for associating tasks with particular CPUs or groups of CPUs known as dispatching domains (167, 171, 278). There is an associated change to group budgets which were introduced in Ada 2005 (169). These changes also concern Annex D.

Other changes concerning budgets relate to the time spent in interrupt handlers (170). In some systems it may be possible to account for time spent in individual interrupts whereas in others it might only be possible to account for time spent in interrupts as a whole. Again this concerns Annex D.

The definition of Volatile is updated to take account of multiprocessors (117).

Finally, there are changes to the core language regarding synchronized interfaces and requeue (30, 215).

2 Scheduling

Ada 83 was remarkably silent about the scheduling of tasks. It muttered about tasks being implemented on multiprocessors or using interleaved execution on a single processor. But it said nothing about how such interleaving might be achieved. It also indicated that a single Ada task might be implemented using several actual processors if the effect would be the same.

Ada 83 introduced the pragma Priority and stated

> if two task with different priorities are both eligible for execution ... then it cannot be the case that the task with the lower priority is executing while the task with the higher priority is not.

The Rationale for Ada 83 says that this rule requires preemptive scheduling. But it says nothing about what happens if several tasks have the same priority. It does however have a dire warning

> Priorities are provided as a tool for indicating relevant degrees of urgency and on no account should their manipulation be used as a technique for attempting to obtain mutual exclusion.

So, apart from the existence of priorities, implementations were free to use whatever scheduling algorithms they liked such as Round Robin time slicing or simply running until blocked.

There was also a bit of a mystery about the delay statement. On the one hand Ada 83 says

suspends execution of the task for at least the duration specified.

The words "at least" caused much confusion. The intent was simply a reminder that a task might not get the processor back at the end of the interval because another task might have become eligible for execution meanwhile. It did not mean that the implementation could willy-nilly delay execution for a longer time.

Another mystery surrounded the meaning of

```
delay 0.0;
```

Ada 83 did state that delay with a negative value is equivalent to a delay statement with a zero value. But it did not say what a delay with a zero value meant. The Rationale remained mute on the topic as well.

However, a general convention seemed to arise that **delay** 0.0; indicated that the task was willing to relinquish the processor and so force a scheduling point.

Ada 95 brought some clarity to the situation in the new Real-Time Systems annex by introducing the pragma Task_Dispatching_Policy and the standard argument of FIFO_Within_Priorities. But the core language did not clarify the effect of a delay of zero. It does say that a delay causes a task to be blocked but if the expiration time has already passed, the task is not blocked. So clearly a negative delay does not block. However, it still has the note that a negative delay is equivalent to delay zero so we could deduce that delay zero does not block and so cannot force scheduling.

But help is at hand in the Real-Time Systems annex where it clearly states that even if a delay does not result in blocking, nevertheless the task goes to the end of the ready queue for its active priority. But that is only for the standard policy of FIFO_Within_Priorities. If a malevolent vendor introduces a curious policy called perhaps Dodgy_Scheduling then it need not follow this rule.

Ada 2005 added further policies namely

```
Non_Preemptive_FIFO_Within_Priorities

Round_Robin_Within_Priorities

EDF_Across_Priorities
```

In the case of Non_Preemptive_FIFO_Within_Priorities a non-blocking delay also sends the task to the end of the ready queue for its active priority. However, a non-blocking delay has absolutely no effect in the case of Round_Robin_Within_Priorities and EDF_Across_Priorities.

The introduction of non-preemptive dispatching revealed a shortcoming that is cured in Ada 2012. The problem is that in such a system there is a need to be able to indicate that a task is willing to be preempted by a task of a higher priority but not by one of the same priority. So somehow we need to say Yield_To_Higher.

Moreover, some felt that it was time to get rid of this strange habit of writing **delay** 0.0; to indicate a scheduling point. Those restricted to the Ravenscar profile, had been forced to write something really gruesome such as

```
delay until Ada.Real_Time.Time_First;
```

Accordingly, the procedure Yield is added to the package Ada.Dispatching so that it becomes

```
package Ada.Dispatching is
   pragma Preelaborate(Dispatching);
   procedure Yield;
   Dispatching_Policy_Error: exception;
end Ada.Dispatching;
```

Calling Yield is exactly equivalent to **delay** 0.0; and similarly causes a bounded error if called from within a protected operation.

There is also a new child package thus

```
package Ada.Dispatching.Non_Preemptive is
   pragma Preelaborate(Non_Preemptive);
   procedure Yield_To_Higher;
   procedure Yield_To_Same_Or_Higher renames Yield;
end Ada.Dispatching.Non_Preemptive;
```

Calling Yield_To_Higher provides the additional facility required for non-preemptive scheduling. Note that, unlike Yield, it can be called from within a protected operation and does not cause a bounded error.

The pedantic programmer can call the precisely named Yield_To_Same_Or_Higher which simply renames Yield in the parent package.

Incidentally, note that since Yield has a side effect, Ada.Dispatching has been downgraded to preelaborable whereas it was pure in Ada 2005.

We now turn to consider an interaction between suspension objects introduced in Ada 95 and EDF scheduling introduced in Ada 2005.

Remember that suspension objects are manipulated by the following package

```
package Ada.Synchronous_Task_Control is
   type Suspension_Object is limited private;
   procedure Set_True(S: in out Suspension_Object);
   procedure Set_False(S: in out Suspension_Object);
   function Current_State(S: Suspension_Object)
                                          return Boolean;
   procedure Suspend_Until_True
                              (S: in out Suspension_Object);
private
   ...
end Ada.Synchronous_Task_Control;
```

The state of a suspension object can be set by calls of Set_True and Set_False. The key feature is that the procedure Suspend_Until_True enables a task to be suspended until the suspension object is set true by some other task. Thus this provides a neat mechanism for signalling between tasks.

Earliest Deadline First (EDF) scheduling is manipulated by the following child package of Ada.Dispatching introduced in Ada 2005 (with use clauses added to save space)

```
with Ada.Real_Time; with Ada.Task_Identification;
use Ada.Real_Time; use Ada.Task_Identification;
package Ada.Dispatching.EDF is
  subtype Deadline is Ada.Real_Time.Time;
  Default_Deadline: constant Deadline := Time_Last;

  procedure Set_Deadline(D: in Deadline;
                         TT: in Task_Id := Current_Task);
  procedure Delay_Until_And_Set_Deadline(
                         Delay_Until_Time: in Time;
                         Deadline_Offset: in Time_Span);
  function Get_Deadline(T: Task_Id := Current_Task)
                              return Deadline;
end Ada.Dispatching.EDF;
```

The procedure Delay_Until_And_Set_Deadline is the key feature. It enables a task to be blocked until the time given by the parameter Delay_Until_Time and sets the deadline so that it is Deadline_Offset after that.

But what is missing in Ada 2005 is the ability for a sporadic task triggered by a suspension object to have its deadline set in a similar manner. This is remedied in Ada 2012 by the addition of the following child package

```
with Ada.Real_Time;
package Ada.Synchronous_Task_Control.EDF is
  procedure Suspend_Until_True_And_Set_Deadline(
                         S: in out Suspension_Object;
                         TS: in Ada.Real_Time.Span);
end Ada.Synchronous_Task_Control.EDF;
```

This enables a task to be blocked until the suspension object S is set true; it then becomes ready with a deadline of Ada.Real_Time.Clock + TS.

The other new feature concerning scheduling in Ada 2012 is the addition of a package Ada.Synchronous_Barriers. This enables many tasks to be blocked and to be released together.

The rationale for needing this facility is explained in the AI concerned. As general purpose computing is moving to parallel architectures and eventually to massively parallel machines, there is a need to efficiently schedule many tasks using barrier primitives. The POSIX OS interface provides a barrier primitive where *N* tasks wait on a barrier and are released simultaneously when all are ready to execute.

There are many situations where the release of *N* tasks is required to execute an algorithm in parallel. Often the calculation is relatively small for each task on each iteration but the number of tasks is relatively high. As an example consider the solution of partial differential equations where one task is allocated to each node of a grid; there might easily be several thousand nodes. Such an example is outlined in [2]. The cost of linearly scheduling and releasing them could remove almost all gains made through parallelization in the first place.

The new package is

```
package Ada.Synchronous_Barriers is
  pragma Preelaborate(Synchronous_Barriers);

  subtype Barrier_Limit is
                range 1 .. implementation-defined;
  type Synchronous_Barrier
          (Release_Threshold: Barrier_Limit) is
                                   limited private;
  procedure Wait_For_Release(
                The_Barrier: in out Synchronous_Barrier;
                Notified: out Boolean);
private
  ...
end Ada.Synchronous_Barriers;
```

The type Synchronous_Barrier has a discriminant whose value indicates the number of tasks to be waited for. When an object of the type is declared its internal counter is set to zero. Thus we might write

```
SB: Synchronous_Barrier(Release_Threshold => 100);
```

When a task calls the procedure Wait_For_Release thus

```
Wait_For_Release(SB, My_Flag);
```

then the task is blocked and the internal counter in SB is incremented. If the counter is then equal to the release threshold for that object (100 in this example), then all the tasks are released. Just one task will have the parameter Notified set to true (the mechanism for selecting the chosen task is not defined). This specially chosen task is then expected to do some work on behalf of all the others. Typically all the tasks will be of the same task type so the code of that type might have

```
Wait_For_Release(SB, My_Flag);
if My_Flag then          -- Gosh, I am the chosen one
  ...                    -- do stuff
end if;
```

Once all the tasks are released, the counter in SB is reset to zero so that the synchronous barrier can be used again.

Care is needed regarding finalization, aborting tasks and other awkward activities. For example, if a synchronous barrier is finalized, then any tasks blocked on it are released and Program_Error is raised at the point of the call of Wait_For_Release.

Many embedded real-time programs, such as those conforming to the Ravenscar profile, run forever. However, there are soft multitasking programs which are hosted on systems such as Windows or Linux and these require closing down in an orderly manner. There are also programs that have mode changes in which the set of tasks involved can be changed dramatically. In such situations it is important that synchronous barriers are finalized neatly.

3 Multiprocessors

In recent years the cost of processors has fallen dramatically and for many applications it is now more sensible to use several individual processors rather than one high performance processor.

Moreover, society has got accustomed to the concept that computers keep on getting faster. This makes them applicable to more and more high volume but low quality applications. But this cannot go on. The finite value of the velocity of light means that increase in processor speed can only be achieved by using devices of ever smaller size. But here we run into problems concerning the nonzero size of Planck's constant. When devices get very small, quantum effects cause problems with reliability.

No doubt, in due course, genuine quantum processors will emerge based perhaps on attributes such as spin. But meanwhile, the current approach is to use multiprocessors to gain extra speed.

One special feature of Ada 2012 aimed at helping to use multiprocessors is the concept of synchronous barriers which were described above. We now turn to facilities for generally mapping tasks onto numbers of processors.

The key feature is a new child package of System thus

```
package System.Multiprocessors is
   pragma Preelaborate(Multiprocessors);

   type CPU_Range is range 0 .. implementation-defined;
   Not_A_Specific_CPU: constant CPU_Range := 0;
   subtype CPU is CPU_Range
                              range 1 .. CPU_Range'Last;

   function Number_Of_CPUs return CPU;
end System.Multiprocessors;
```

Note that this is a child of System rather than a child of Ada. This is because System is generally used for hardware related features.

Processors are given a unique positive integer value from the subtype CPU. This is a subtype of CPU_Range which also includes zero; zero is reserved to mean not allocated or unknown and for clarity is the value of the constant Not_A_Specific_CPU.

The total number of CPUs is determined by calling the function Number_Of_CPUs. This is a function rather than a constant because there could be several partitions with a different number of CPUs on each partition. And moreover, the compiler might not know the number of CPUs anyway.

Since this is not a Remote Types package, it is not intended to be used across partitions. It follows that a CPU cannot be used by more than one partition. The allocation of CPU numbers to partitions is not defined; each partition could have a set starting at 1, but they might be numbered in some other way.

Tasks can be allocated to processors by an aspect specification. If we write

```
task My_Task
   with CPU => 10;
```

then My_Task will be executed by processor number 10. In the case of a task type then all tasks of that type will be executed by the given processor. The expression giving the processor for a task can be dynamic.

Moreover, in the case of a task type, the CPU can be given by a discriminant. So we can have

```
task type Slave(N: CPU_Range)
   with CPU => N;
```

and then we can declare

```
Tom: Slave(1);
Dick: Slave(2);
Harry: Slave(3);
```

and Tom, Dick and Harry are then assigned CPUs 1, 2 and 3 respectively. We could also have

```
Fred: Slave(0);
```

and Fred could then be executed by any CPU since 0 is Not_A_Specific_CPU.

The aspect can also be set by a corresponding pragma CPU. (This is an example of a pragma born obsolescent as explained in the paper on contracts and aspects.) The aspect CPU can also be given to the main subprogram in which case the expression must be static.

Further facilities are provided by the child package System.Multiprocessors.Dispatching_Domains as shown below. Again we have added use clauses to save space and also have often abbreviated Dispatching_Domain to D_D.

```
with Ada.Real_Time; with Ada.Task_Identification;
use Ada.Real_Time; use Ada.Task_Identification;
package System.Multiprocessors.Dispatching_Domains is
   pragma Preelaborate(Dispatching_Domains);

   Dispatching_Domain_Error: exception;

   type Dispatching_Domain(<>) is limited private;
   System_Dispatching_Domain: constant D_D;

   function Create(First, Last: CPU) return D_D;
   function Get_First_CPU(Domain: D_D) return CPU;
   function Get_Last_CPU(Domain: D_D) return CPU;
   function Get_Dispatching_Domain(
                  T: Task_Id := Current_Task) return D_D;

   procedure Assign_Task(
         Domain: in out Dispatching_Domain;
         CPU: in CPU_Range := Not_A_Specific_CPU;
         T: in Task_Id := Current_Task);

   procedure Set_CPU(CPU: in CPU_Range;
                  T: in Task_Id := Current_Task);

   function Get_CPU(T: in Task_Id := Current_Task)
                                 return CPU_Range;

   procedure Delay_Until_And_Set_CPU(
                  Delay_Until_Time: in Time;
                  CPU: in CPU_Range);
private
   ...
end System.Multiprocessors.Dispatching_Domains;
```

The idea is that processors are grouped together into dispatching domains. A task may then be allocated to a

domain and it will be executed on one of the processors of that domain.

Domains are of the type Dispatching_Domain. This has unknown discriminants and consequently uninitialized objects of the type cannot be declared. But such an object can be initialized by the function Create. So to declare My_Domain covering processors from 10 to 20 inclusive we can write

```
My_Domain: Dispatching_Domain := Create(10, 20);
```

All CPUs are initially in the System_Dispatching_Domain. A CPU can only be in one domain. If we attempt to do something silly such as create overlapping domains by for example also writing

```
My_Domain_2: Dispatching_Domain := Create(20, 30);
```

then Dispatching_Domain_Error is raised because in this case, CPU number 20 has been assigned to both My_Domain and My_Domain_2.

The environment task is always executed on a CPU in the System_Dispatching_Domain. Clearly we cannot move all the CPUs from the System_Dispatching_Domain other wise the environment task would be left high and dry. Again an attempt to do so would raise Dispatching_Domain_Error.

A very important rule is that Create cannot be called once the main subprogram is called. Moreover, there is no operation to remove a CPU from a domain once the domain has been created. So the general approach is to create all domains during library package elaboration. This then sets a fixed arrangement for the program as a whole and we can then call the main subprogram.

Each partition has its own scheduler and so its own set of CPUs, dispatching domains and so on.

Tasks can be assigned to a domain in two ways. One way is to use an aspect

```
task My_Task
  with Dispatching_Domain => My_Domain;
```

If we give both the domain and an explicit CPU thus

```
task My_Task
  with CPU => 10, Dispatching_Domain => My_Domain;
```

then they must be consistent. That is the CPU given must be in the domain given. If it is not then task activation fails (hands up all those readers who thought it was going to raise Dispatching_Domain_Error). If for some reason we write

```
task My_Task
  with CPU => 0, Dispatching_Domain => My_Domain;
```

then no harm is done. Remember that there is not a CPU with number zero but zero simply indicates Not_A_Specific_CPU. In such a case it would be better to write

```
task My_Task
  with CPU => Not_A_Specific_CPU,
       Dispatching_Domain => My_Domain;
```

The other way to assign a task to a domain is by calling the procedure Assign_Task. Thus the above examples could be written as

```
Assign_Task(My_Domain, 10, My_Task'Identity);
```

giving both domain and CPU, and

```
Assign_Task(My_Domain, T => My_Task'Identity);
```

which uses the default value Not_A_Specific_CPU for the CPU.

Similarly, we can assign a CPU to a task by

```
Set_CPU(A_CPU, My_Task'Identity);
```

Various checks are necessary. If the task has been assigned to a domain there is a check to ensure that the new CPU value is in that domain. If this check fails then Dispatching_Domain_Error is raised. Of course, if the new CPU value is zero, that is Not_A_Specific_CPU then it simply means that the task can then be executed on any CPU in the domain.

To summarize the various possibilities, a task can be assigned a domain and possibly a specific CPU in that domain. If no specific CPU is given then the scheduling algorithm is free to use any CPU in the domain for that task.

If a task is not assigned to a specific domain then it will execute in the domain of its activating task. In the case of a library task the activating task is the environment task and since this executes in the System_Dispatching_Domain, this will be the domain of the library task.

The domain and any specific CPU assigned to a task can be set at any time by calls of Assign_Task and Set_CPU. But note carefully that once a task is assigned to a domain other than the system dispatching domain then it cannot be assigned to a different domain. But the CPU within a domain can be changed at any time; from one specific value to another specific value or maybe to zero indicating no specific CPU.

It is also possible to change CPU but for the change to be delayed. Thus we might write

```
Delay_Until_And_Set_CPU(
                Delay_Until_Time => Sometime,
                CPU => A_CPU);
```

Recall we also have Delay_Until_And_Set_Deadline in Ada.Dispatching.EDF mentioned earlier.

Note that calls of Set_CPU and Assign_Task are defined to be task dispatching points. However, if the task is within a protected operation then the change is deferred until the next task dispatching point for the task concerned. If the task is the current task then the effect is immediate unless it is within a protected operation in which case it is deferred as just mentioned. Finally, if we pointlessly assign a task to the system dispatching domain when it is already in that domain, then nothing happens (it is not a dispatching point).

There are various functions for interrogating the situation regarding domains. Given a domain we can find its range of CPU values by calling the functions Get_First_CPU and Get_Last_CPU. Given a task we can find its domain and CPU by calling Get_Dispatching_Domain and Get_CPU. If a task is not assigned a specific CPU then Get_CPU naturally returns Not_A_Specific_CPU.

In order to accommodate interrupt handling the package Ada.Interrupts is slightly modified and now includes the following function

```
function Get_CPU(Interrupt: Interrupt_Id)
        return Systems.Multiprocessors.CPU_Range;
```

This function returns the CPU on which the handler for the given interrupt is executed. Again the returned value might be Not_A_Specific_CPU.

The Ravenscar profile is now defined to be permissible with multiprocessors. However, there is a restriction that tasks may not change CPU. Accordingly the definition of the profile now includes the following restriction

```
No_Dependence =>
        System.Multiprocessors.Dispatching_Domains
```

In order to clarify the use of multiprocessors with group budgets the package Ada.Execution_Time.Group_Budgets introduced in Ada 2005 is slightly modified. The Ada 2005 version is

```
with System;
package Ada.Execution_Time.Group_Budgets is

  type Group_Budget is tagged limited private;

...type Group_Budget_Handler is access
      protected procedure (GB: in out Group_Budget);

... ...          -- and so on
private
  ...
end Ada.Execution_Time.Group_Budgets;
```

However, in Ada 2012 the type Group_Budget has a discriminant giving the CPU thus

```
type Group_Budget(
            CPU: System.Multiprocessors.CPU :=
              System.Multiprocessors.CPU'First)
                    is tagged limited private;
```

This means that a group budget only applies to a single processor. If a task in a group is executed on another processor then the budget is not consumed. Note that the default value for CPU is CPU'First which is always 1.

4 Interrupt timers and budgets

It will be recalled that Ada 2005 introduced three packages for monitoring the CPU time used by tasks. They are a root package Ada.Execution_Time plus two child packages thus

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, 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 declared in the root package Ada.Execution_Time.

However, it was left implementation defined in Ada 2005 as to how the time spent in interrupts was to be accounted. The Ada 2005 RM says

> It is implementation defined which task, if any, is charged the execution time that is consumed by interrupt handlers and run-time services on behalf of the system.

As noted in the AI, a common and simple implementation will charge the time consumed by the interrupt handlers to the task executing when the interrupt is generated. This is done under the assumption that the effect of interrupt handlers on the execution time clocks is negligible since the interrupt handlers are usually very short pieces of code. However, in real-time systems that undertake an intensive use of interrupts, this assumption may not be realistic. For example, Ada 2005 introduced timed events that can execute handlers in interrupt context. The facility is convenient and has low overheads, and therefore programmers are tempted to put more code into these handlers.

It is thus considered important to be able to measure time spent in interrupts and so facilities to do this are added in Ada 2012.

The root package is extended by the addition of two Boolean constants, Interrupt_Clocks_Supported and Separate_Interrupt_Clocks_Supported, and also a function Clocks_For_Interrupts so in outline it becomes

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Real_Time;  use Ada.Real_Time;
package Ada.Execution_Time is

  type CPU_Time is private;

  ...

  function Clock(T: Task_Id := Current_Task)
                                    return CPU_Time;

  ...

  Interrupt_Clocks_Supported:
          constant Boolean := implementation-defined;
  Separate_Interrupt_Clocks_Supported:
          constant Boolean := implementation-defined;

  function Clocks_For_Interrupts return CPU_Time;
```

```
   private
      ... -- not specified by the language
   end Ada.Execution_Time;
```

The constant Interrupt_Clocks_Supported indicates whether the time spent in interrupts is accounted for separately from the tasks and then Separate_Interrupt_Clocks_Supported indicates whether the time is accounted for each interrupt individually.

The new function Clocks_For_Interrupts returns the CPU_Time used over all interrupts. It is initialized to zero.

Time accounted for in interrupts is not also accounted for in individual tasks. In other words there is never any double accounting.

Calling the function Clocks_For_Interrupts if Interrupt_Clocks_Supported is false raises Program_Error. Note that the existing function Clock has a parameter giving the task concerned whereas Clocks_For_Interrupts does not since it covers all interrupts.

A new child package of Ada.Execution_Time is provided for monitoring the time spent in individual interrupts. Note that this package always exists even if the Boolean constant Separate_Interrupt_Clocks_Supported is false. Its specification is

```
   package Ada.Execution_Time.Interrupts is
      function Clock(Interrupt: Ada.Interrupts.Interrupt_Id)
                                              return CPU_Time;
      function Supported(
                        Interrupt: Ada.Interrupts.Interrupt_Id)
                                              return Boolean;
   end Ada.Execution_Time.Interrupts;
```

The function Supported indicates whether the time for a particular interrupt is being monitored. If it is then Clock returns the accumulated CPU_Time spent in that interrupt handler (otherwise it returns zero). However, if the overall constant Separate_Interrupt_Clocks_Supported is false then calling this function Clock for any particular interrupt raises Program_Error.

The package Ada.Execution_Time.Timers is exactly the same in Ada 2012. However, as mentioned earlier, the package Ada.Execution_Time.Group_Budgets is now defined to work on a single processor and the type Group_Budget is modified to include a discriminant giving the CPU concerned.

5 Volatile

This is a curious topic and created much debate. For the collector of statistics the real part of the AI is less than two pages but the appendix has nearly twenty pages of chatter!

The problem is all about sharing variables and ensuring that things happen in the correct order. Moreover, we need to avoid the overhead of protected objects particularly on microprocessors where we might be using low level features such as memory barriers discussed in Section 2 above.

Suppose we have two tasks A and B which access some shared data perhaps in a nice package Common thus

```
   package Common is
      ...
      Data: Integer;
      pragma Volatile(Data);
      Flag: Boolean;
      pragma Volatile(Flag);
      ...
   end Common;
```

and in task A we write

```
   with Common; use Common;
   task A is
      ...
      Data := 42;
      Flag := True;
      ...
   end A;
```

whereas in task B we have

```
   with Common; use Common;
   task B is
      Copy: Integer;
   begin
      ...
      loop
         exit when Flag;      -- spin
      end loop;
      Copy := Data;
      ...
   end B;
```

The idea is that task A assigns some value to Data and then indicates this to task B by setting Flag to true. Meanwhile, task B loops checking Flag and when it is found to be true, then reads the Data.

Does this work in Ada 2005? Hmm. Nearly. There are three things that need to be ensured. One is that Flag gets changed in one lump. Another is that the new value of Data assigned by task A truly is updated when task B reads it. And the third is that the actions happen sequentially. Well, we should have applied pragma Atomic to Flag to ensure the first but since it is of type Boolean we might get away with it. And note that Atomic implies Volatile anyway. Also Atomic ensures that the actions are sequential.

So, with the pragma Volatile changed to Atomic for Flag, it does indeed work in Ada 2005 because Volatile ensures that read and writes are to memory and so things do happen in the correct order. However, this is overkill. It is not necessary that all accesses are to memory; all that matters is that they happen in the correct order so they could be to some intermediate cache. Indeed, there might be nested caches and as hardware evolves it is becoming more difficult to make general statements about its structure; hence we can really only make statements about the effect.

The possibility of introducing a new pragma Coherent was debated for some time. However, it was ultimately

concluded that the definition of Volatile should be weakened. In Ada 2005 it says

> For a volatile object all reads and updates of the object as a whole are performed directly to memory.

In Ada 2012 it says

> All tasks of the program (on all processors) that read or write volatile variables see the same order of updates to the variables.

Of course, in Ada 2012, we use aspects so the package Common becomes

```
package Common is
  ...
  Data: Integer
    with Volatile;
  Flag: Boolean
    with Atomic;          -- Atomic implies Volatile
  ...
end Common;
```

where we have given Atomic for Flag. As mentioned above, Atomic implies Volatile so it is not necessary to give both. However, if we do have to give two aspects, it is much neater that the one aspect specification does this whereas two distinct pragmas would be necessary.

It is said that this change brings the meaning of volatile into line with that in C. However, it has also been said that the definition of volatile in C is unclear.

6 Synchronized interfaces and requeue

Ada 2005 introduced interfaces of various kinds: limited, nonlimited, synchronized, task, and protected. These form a hierarchy and in particular task and protected interfaces are forms of synchronized interfaces. The essence of this was to integrate the OO and real-time features of Ada. But a problem was discovered regarding requeue as described in a paper presented at IRTAW 2007 [3].

Some examples of interfaces will be found in [2] or [4] where various implementations of the readers and writers paradigm are explained.

The operations of a synchronized interface are denoted by subprograms. Thus we might have

```
package Pkg is
  type Server is synchronized interface;
  procedure Q(S: in out Server; X: in Item) is abstract;
end Pkg;
```

We can then implement the interface by a task type or by a protected type. This introduces several different ways of implementing the operation Q. It can be by an entry, or by a protected procedure or by a normal procedure. For example using a task type we might have

```
package TP1 is
  task type TT1 is new Server with
                -- Q implemented by entry
    entry Q(X: in Item);
  end TT1;
end TP1;
```

or

```
package TP2 is
  task type TT2 is new Server with
                -- Q implemented by a normal procedure
  end TT2;
  procedure Q(S: in out TT2; X: in Item);
end TP2;
```

Similarly using a protected type we might have

```
package PP1 is
  protected type PT1 is new Server with
                -- Q implemented by entry
    entry Q(X: in Item);
    ...
  end PT1;
end PP1;
```

or

```
package PP2 is
  protected type PT2 is new Server with
                -- Q implemented by a protected procedure
    procedure Q(X: in Item);
    ...
  end PT2;
end PP2;
```

or

```
package PP3 is
  protected type PT3 is new Server with
                -- Q implemented by a normal procedure
    ...
  end PT3;
  procedure Q(X: in out PT3; X: in Item);
end PP3;
```

So the interface Server could be implemented in many different ways. And as usual we could dispatch to any of the implementations. We could have

```
Server_Ptr: access Server'Class := ...
...
Server_Ptr.Q(X => An_Item);
```

and this will dispatch to the implementation of Q concerned.

So a call of Q could end up as a call of an entry in a task, an entry in a protected object, a protected procedure in a protected object, or an ordinary procedure.

Two curious situations arise. One concerns timed calls. We could write a timed call such as

```
select
  Server_Ptr.Q(An_Item);
or
  delay Seconds(10);
end select;
```

and this will always be acceptable. It will dispatch to the appropriate operation. If it is an entry then it will be a timed call. But if it is not an entry then no time-out is possible and so by default the call will always go ahead.

The other curious situation concerns requeue. In this case there is no obvious default action. It is not possible to requeue a procedure call since there is no queue on which to hang it.

The first proposal to do something about this was simply not to allow requeue at all on interfaces. And indeed this was the solution adopted in Ada 2005.

However, this is not really acceptable as explained in [3]. The next idea was to raise some exception if it turned out that the destination was not an entry. But this was considered unsatisfactory.

So it was concluded that if we do a requeue then it must be statically checked that it will dispatch to an entry so that the requeue is possible. The next proposal was that there should be a pragma Implemented giving requirements on the operation. Thus we might have

```
procedure Q(S: in out Server; X: in Item) is abstract;
pragma Implemented(Q, By_Entry);
```

and the compiler would ensure that all implementations of the interface Server did indeed implement Q by an entry so that requeue would always work. The other possible values for the pragma were By_Protected_Procedure and By_Any.

The world changed when the notion of an aspect was invented and so after much discussion the final solution is that we there is now an aspect Synchronization so we write

```
procedure Q(S: in out Server; X: in Item) is abstract
   with Synchronization => By_Entry;
```

and we are now assured that we are permitted to do a requeue on Q for any implementation of Server. The other possible values for the aspect Synchronization are By_Protected_Procedure and Optional.

In summary, if the property is By_Entry then the procedure must be implemented by an entry, if the property is By_Protected_Procedure then the procedure must be implemented by a protected procedure, and if the property is Optional then it can be implemented by an entry, procedure or protected procedure. Naturally enough, the aspect cannot be given for a function.

There are a number of rules regarding consistency. The aspect Synchronization can be applied to a task interface or protected interface as well as to a synchronized interface. However, if it is applied to a task interface then the aspect cannot be specified as By_Protected_Procedure for obvious reasons.

If a type or interface is created by inheritance from other interfaces then any Synchronization properties are also inherited and must be consistent. Thus if one is By_Entry then the others must also be By_Entry or Optional.

A final minor improvement mentioned in the Introduction concerns renaming. Since the days of Ada 83 it has been possible to rename an entry as a procedure thus

```
procedure Write(X: in Item) renames Buffer.Put;
```

where Put is an entry in a task Buffer. But in Ada 83 it was not possible to do a timed call using Write. This was corrected in Ada 2005 which allows a timed call on a renaming.

Similarly, when requeue was introduced in Ada 95, it was not possible to do a requeue using Write. This anomaly is corrected in Ada 2012. So now both timed calls and requeue are permitted using a renaming of an entry.

References

[1] ISO/IEC JTC1/SC22/WG9 N498 (2009) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of Amendment 2 to ISO/IEC 8652*.

[2] J. Barnes (2006), *Programming in Ada 2005*, Addison-Wesley.

[3] A. Burns and A. Wellings (2007), *Integrating OOP and Tasking – the missing requeue*, from IRTAW 2007,www.ada-auth.org/ai-files/grab_bag/requeue.pdf.

[4] J. Barnes (2008), *Ada 2005 Rationale*, LNCS 5020, Springer-Verlag.

What is Language Technology in Our Time

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

Abstract

For a language that had its trademark in safety and robustness, discipline and control, in the last 20 years, Ada has steadily extended its wealth of features and capabilities to a considerable extent, yet within the bounds of its original mission. However, perhaps because the industrial systems written in Ada are unlike to evolve just to catch up on new features, the pace of advancement in the Ada language risks being faster than that of its users. Traditional education and training are not going to bridge the gap, because – for reasons that escape common sense – Ada is often not in the software engineering curriculum and the industrial economics leave little room for training. Arguably, it should be the language to reach out to prospective users, more than the reverse. But this may need "the language" to encompass more than a programming language does in the traditional sense of the classic compiler-debugger pair. It may be libraries, patterns, frameworks, tutorials, and many other elements that one way or another seem to belong in the general concern of language technology in our time, and make the fortune of far less solid languages.

Why this panel

Not a language designer by education, I have had the good fortune of taking part in the technical and procedural activities that have given shape to the latest 20 years of Ada evolution. Not that I had carried out design or implementation activities myself. The way that work was carried out made room for one technical lead, and a number of equally competent experts, who were there to discuss the wisdom of the language features being proposed, their impact on backward compatibility, their complexity in both implementation and use. Luckily in fact, there was also room for a few other people with much more vertical skills, who were there to represent user community needs, requirements and proposals, in the (often tentative) way of features and semantics. I have been part of the latter segment of the group and I have witnessed in awe the skills it takes to shape a programming language and to direct its evolution steadily and consistently.

Having a tendency to the philosophical side of things, over the years I have grown deep in the belief that a programming language is a view of the world that it aims to represent (well, I should in fact say, design and implement rather than represent, but I think you have caught my drift). Perhaps, when the language sees itself as general-purpose, then I should replace "is" by "contains" on account of the possible presence of multiple "worlds" in that language universe; yet this causes no fundamental change – I think – to the point that I am trying to make.

The reader should be advised that when I am saying language, I am not especially interested in its syntax (which is of course important in a number of ways, but not to my argument here), but in what you would call "expressive power": what you can say with it, not how you do it.

Wittgenstein said: "All I know is what I have words for". Following that, what the language allows saying exists and what it can't (fully or rightly) just doesn't. This is why I said that a language is a view of some world: because it allows you to express your part of that world, within the large yet finite set of entities that live in the language ontology, but nothing proper (hence just nothing) outside of it.

This observation – or claim if you will – is what caused me to propose this panel. Let me explain you why. If a programming language is a view of some world, then it must be that the language designer is the primary owner of that view and the technology that puts the expressive power of that language into existence should conform to that world view and project it toward the user. What do I mean when I say that language technology should conform to the world view of the language? I wish to convey that the language technology should help the programmer "see" the concepts that one can express, whether simple or articulate, so that you can use them for your own purpose, in the way the language design intended them to.

My question is – actually, has long been – how can this ever be done by language technology? What must the language technology include to that end?

As a side digression, you will note a corollary of my argument so far: if a programming language is what I have just tried to explain, what is language teaching? Probably, it should be more a description of the world entities and the world rules that the language expresses (I am tempted to call all of this collectively, the world architecture according to the language) than most it is taught today, which wanders between mustering syntax (so that you can speak the language without necessarily understanding it) and general abstract principles that perhaps preside over numerous world architectures, but are not any one particular instance of it.

Of course at the basic level of language technology you have a compiler and a debugger. You don't go far without a compiler, because it obviously is the compiler that makes the language exist outside of the mind of the language designer. Of course, novices also need a debugger to help them make some sense of their errors. And experts occasionally need debuggers too, though normally for more sophisticated purposes.

Surely however, the world architecture according to the language is not presented to the user from the compiler. Where from then? From the language manual? Perhaps so, but only a long time ago, when the language view of the world was small and simple and there was a lesser gap between the language elements and the world ontology that they expressed. No single user today would use a language manual to learn the world view of a language. In fact, perhaps inevitably, language manuals have progressively transmogrified into legalistic instruments intended for developers of the language technology; less and less, and finally no longer for users. There undoubtedly is a large gap to fill there. And the question I am asking is how and by what means this can be filled. To help answer that question we should study what factors determine the success of programming languages, aside from hype and gravitational force of conformism. One of the answers is surely that language technology in those cases has taken a large and comprehensive connotation: it is libraries, patterns, frameworks, tutorials, and many other immaterial elements that contribute to proclaiming and divulging the world view of the language, and draw the user into it, by force of ease, evidence and intellectual strength.

The purpose of this panel – in my original intent – is to ask three leading figures, all of whom professionally confronted with the challenge I am posing, what do they see language technology in our time to be. I have sought the opinion of a language designer, of a language implementer, and of a language educator. I have of course my own opinion, and I will fiercely voice it in the panel, but the opinions of front-liners have to be heard first. For this reason this text just serves as an introduction to the position statements offered by the panellists.

Thoughts on Ada and Language Technology in Our Time

Franco Gasperoni
AdaCore, gasperoni@adacore.com

It is fascinating to open the second edition of John Barnes' book "Programming in Ada". If you are lucky to own a copy I encourage you to read the "Foreword" by Jean Ichbiah, the two prefaces, section 1.1 "History", and the last page in the "Finale". John's witty style makes the reading very enjoyable. Upon completing the reading, almost 20 years after my initial one, I emerged with a number of thoughts described in the following pages that could be summarized as follows:

- The first version of Ada (1983) was a language that was way ahead of its time. Furthermore, Ada suffered from the association with the US Department of Defense whose reputation had been tarnished by the Vietnam War.
- There are, broadly speaking, four target domains for computing applications: casual programming, enterprise software, mission-critical systems, mobile apps for all kinds and shapes. A programming language cannot be successful in all of these domains.Ada is designed for mission-critical industrial systems where it has a strong track record. In this context we observe the following challenges for today's programming languages:
  - Modeling and "qualified" components;
  - Security;
  - Seamless programming of multi and many-core machines.
- Ada 2012 and its follow-ons have many assets to address the above challenges and continue playing a key role in mission-critical systems of the 21st century.
- The future of any language lies in the young generations of programmers. To be vibrant the Ada experience must be readily available to digital natives.

"Programming in Ada" by John Barnes, 2nd Edition, October 1983

Here are some interesting excerpts from John's book:

From the Foreword by Jean Ichbiah: "*Here is a major contradiction in any design work. On the one hand, one can only reach an harmonious integration of several features by immersing oneself into the logic of the existing parts; it is only in this way that one can achieve a perfect combination. On the other hand, this perception of perfection, and the implied acceptance of certain unconscious assumptions, will prevent further progress.*"

From the preface to the first edition: "*This book is about Ada, the new and powerful programming language originally developed on behalf of the US Department of Defense, for use in embedded systems. Typical of such systems are those of process control, missile guidance or even the sequencing of a dishwasher. […] Although originally intended for embedded systems, it is a general purpose language and could, in time, supersede FORTRAN and even COBOL.*"

From Section 1.1 "History" in the "Introduction": *"The story of Ada goes back to about 1974 when the United States Department of Defense realized that it was spending far too much on software. It carried out a detailed analysis of how its costs were distributed over the various application areas and discovered that over half of them were directly attributed to embedded systems.*

Further analysis was directed towards the programming languages in use in the various areas. It was discovered that COBOL was the universal standard for data processing and FORTRAN was a similar standard for scientific and engineering computation. Although these languages were not modern, the fact that they were uniformly applied in their respective areas meant that unnecessary and expensive duplication was avoided.

The situation with regard to embedded systems was however quite different. The number of languages in use was enormous. Not only did each of the three Armed Services have their own favorite high level languages, but they also used many assembly languages as well. Moreover, the high level languages had spawned variants. It seemed that successive contracts had encouraged the development of special versions aimed at different applications. The net result was that a lot of money was being spent on an unnecessary number of compilers. There were also all the additional costs of training and maintenance associated with a lack of standardization."

Jean Ichbiah

Jean Ichbiah, the principle designer of Ada in the 70s and early 80s, was French born (1940). Grandson of Greek and Turkish immigrants, he was a brilliant student who graduated from elite French colleges. He graduated from the "École Polythechnique" and the "École des Ponts et

Chaussées" one of the best civil engineering schools in France. During his education he developed a strong sense for aesthetics: for Ichbiah both form and function were important (think Eiffel Tower). Moving on from civil engineering he went to MIT in Boston where Ichbiah did a thesis on syntactical analysis of programming languages. Ichbiah's sense for aesthetics and the equal importance of form and function are very much present in Ada and Ada's form is one of the things that makes Ada programs so readable years after the fact. On the less positive side, Ichbiah's former colleagues point out that Jean was a "control freak" and a "nano-manager". As we will see below, this had an impact on the Ada program library concept which was fixed by Richard Stallman and Robert Dewar who are both at the extreme opposite of "control mania" and "nano-management".

Ada and the DoD

The US Department of Defense (DoD) was the sponsor of Ada's design in the 70s and was behind its adoption in the 80's. This was both a boon and a handicap. Because of the cold war era, the DoD had deep pockets. Unfortunately, because of the Vietnam War which ended in the mid-70s, the aura of the DoD wasn't what it used to be. Add to that the DoD mandate of the 80s to use Ada on defense contracts and straight out of the cradle Ada was seen as "un-cool" by the computing undercurrent.

Ada 1.0

Ada 1.0 (also known as Ada 83) was a language that was way too powerful for its time. Compiling it for an 8 bit microcontroller was a daunting task and never really happened until recently when Ada 3.0 (Ada 2005) was made available on Atmel's 8-bit AVR.

In the mid-80s Ada was ahead of what compiler and computer technology could do. This, for instance, lead Alsys (the company founded by Jean Ichbiah), to pioneer the use of virtual memory on x86 PCs and to bundle memory cards with the compiler.

Another way that Ada was ahead of its time is that its users were dealing with defense-related problems of a complexity not generally encountered in the industry until 10-15 years later (remember Ronald Regan's march 23, 1983 speech on the "Strategic Defense Initiative" later known as Star Wars). Ada's designers anticipated this increase in complexity and added mandatory consistency checks at various levels of the software construction chain. Unfortunately, in doing this Ada 1.0 broke the sociology of a fundamental element in collaborative software development. Unlike its 1970s C predecessor, Ada 1.0 had a collaboration bottleneck: the order-of-compilation model and its centralized "program library file" assumption. This is one of those "unconscious assumptions" Jean Ichbiah talks about in the Foreword of John's 1983 book. From the Ada 1.0 reference manual:

> *10.4. The Program Library*
>
> *Compilers are required to enforce the language rules in the same manner for a program consisting of several compilation units (and subunits) as for a program submitted as a single compilation. Consequently, a library file containing information on the compilation units of the program library must be maintained by the compiler or compiling environment. This information may include symbol tables and other information pertaining to the order of previous compilations.*
>
> *A normal submission to the compiler consists of the compilation unit(s) and the library file. The latter is used for checks and is updated for each compilation unit successfully compiled.*

Today, programming languages foster the same sociological "gestalt" for collaborative software development: the sources, nothing-but the sources. Some readers may not understand what I am talking about as this is so obvious today. How could we have lived through a take-your-turn-to-compile-and-if-your-colleague-recompiles-you-may-have-to-recompile-too?
This started from a noble intention and a critical Ada insight: type safety (type-checking) must operate on the overall program, across "compilation units" in Ada's parlance. This was completely novel at the time. What was unfortunate is the approach "à la Colbert1" that was taken in Ada 1.0. In fact, instead of sticking to a purely regulatory philosophy as it was subsequently done in Ada 2.0, Ada 1.0 strongly implied an implementation approach to program-wide type safety. This implied approach was sociologically broken. This unfortunate oversight was fixed in the early 90s thanks to the intervention of Richard Stallman and Robert Dewar. Richard Stallman was adamant that Ada's type-safety-across-the-program rule did not create order of compilation dependencies (a sociological bottleneck). Robert Dewar, leveraging on the evolution of computers, found a way to achieve Ada's overall type safety rule with a pure source-based model. Today we can have the cake and eat it too: C's freedom and Ada's type safety.

Apart from this, Ada 1.0 was visionary on the fundamental properties that a programming language for industrial systems had to possess. Decade after decade these properties pay back their dividends to users of Ada. My favorite property is the ability to communicate to other humans the key elements of what is being computed and have the compiler check their consistent use. This

[1] Jean-Baptiste Colbert served as the Minister of Finances of France under King Louis XIV from 1665 to 1683. Colbert is referenced here because of his doctrine in which the State exerts a strong directive influence on the economy as opposed to a merely regulatory role. His doctrine is also known as "Colbertism" or "dirigism". If you are still wondering why I mentioned Colbert in the context of the Ada 1.0 centralized program library, consider the following. Jean Ichbiah graduated from elite French colleges both pure products of "Colbertism". The Ada 1.0 centralized program library assumption was the approach "à la Colbert" towards ensuring that an Ada program was type safe as a whole.

property has evolved and strengthened from Ada 1.0 (1983), to Ada 2.0 (1995), to Ada 3.0 (2005), to today's Ada 4.0 (2012).

General-Purpose Does Not Mean Universal

When Ada 1.0 came about the DoD was the biggest software contractor. Everything had to be programmed. No spreadsheets. To compute a simple linear regression we had to do it with pencil and paper and a hand calculator. The luckier ones had access to statistical packages on minicomputers or mainframes. Programming languages were the heart of the matter: they were the only way to get anything done with a large, bulky, slow, expensive, unconnected computer. Programming was the realm of mathematicians, physicists, chemists, engineers. A programming language, a dumb editor, and a compiler were all that was available then. No IDEs, no components, no frameworks.

When "Green" was designed, the attitude towards developing an embedded application was: let's do it from scratch using the best possible language, a language that would decrease the chance of writing "wrong" code, a language that would make abstractions clear, a language that would facilitate the reading, use, and re-use of software.

From the mid-50s onwards, computer scientists nurtured the dream that a sound, general-purpose, programming language would be the key to computing salvation. In the 60s general-purpose did not include embedded systems, which were just starting to emerge. IBM's PL/I effort was focused around IBM's concerns of the time and did not have embedded systems in mind. A general-purpose language targeting embedded systems was needed. In the wildest of dreams that language could be used from embedded real-time systems to accounting applications in the DoD.

Ada substantiated the dream that a general-purpose programming language could be used universally to program all computing devices and applications. This idea strengthened throughout the 80s, 90s, and a portion of this century. After placing that hope on Ada 1.0, the community placed its bets on C++ and then Java, hoping to find the programming language that does it all. This never happened.

The message of this section is that we cannot expect a general-purpose programming language, be it Ada, C++, or Java, to be used universally. The spread and usage of a language is correlated with the economics and evolution of the application domain which gave birth to that language. In this respect some languages such as Ada and C++ compete because their application domains overlap, while neither is a serious contender in web-centric applications.

Because there is no "universal language", systems are being written using several idioms and approaches. For this to be viable, languages should be able to talk to each other. Ada realized the importance of this in its 2.0 release and today Ada interfaces well with subsystems written in other languages. In the end what matters is being a good play-mate: if you are, everyone wants to play with you.

Raising the Level of Abstraction: Model It

The myth of a universal programming language is slowly fading (I wrote universal not general-purpose). Universality comes at the cost of expressivity. Imagine doing math, physics, or engineering without mathematical notation. Imagine having to spell everything out in plain English. Sure we can do that. English is a general-purpose (and as close as we can get universal) language. But how expressive is it to talk math, physics, and engineering? Likewise, how can a team of scientists and engineers model a "phenomenon"? What language can the team use to devise that model? The key is in the meaning of the word ontology. From Wikipedia:

"In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain and may be used to describe the domain. In theory, an ontology is a "formal, explicit specification of a shared conceptualization". An ontology renders shared vocabulary and taxonomy which models a domain with the definition of objects and/or concepts and their properties and relations. Ontologies are the structural frameworks for organizing information and are used in … as a form of knowledge representation about the world or some part of it."

Conventional programming languages such as FORTRAN, COBOL, C … have been used to create, express, maintain and evolve the ontology of the "phenomenon" that we want to model. To identify and communicate among humans the patterns of interactions between the elements of the ontology there has been a race to design the "best" high-level general-purpose programming language: Ada, C++, Java …. These languages have grown out of the Church–Turing computability thesis. The Church-Turing thesis tells us that to be processed mechanically a "phenomenon" must be modeled as computable mathematical functions. Although accurate, this view can limit the horizon of our possibilities. In fact, to create a computable model of the "phenomenon" we may want to use human-understandable languages that are not computable.

To raise the level of abstraction beyond general-purpose programming languages we could start from the ontology of the application domain and model the "phenomenon" using the language and symbols that are part of the application domain, i.e. its "natural" ontology. We could use that ontology to design, communicate, and convince others and ourselves that our model of the "phenomenon" is faithful. The last step would be to translate the model into a language that machines can understand. This last step could be done by humans, machines themselves, or a

mixture of both. Welcome back domain-specific languages (DSL) also known as modeling languages (and 4GL before that): UML, AADL, Simulink, Modelica … As a side note, when human intervention is required to go from the model to the machine the use of a high-level programming language such as Ada keeps translation and maintenance costs down.

When the ontology of our "phenomenon" is clearly defined and well established in the application domain, DSL are an attractive complement and even a substitute for general-purpose programming languages. In application domains without an obvious domain-specific language to express the ontology, high-level general-purpose programming languages are the best we have. In fact, attempts at consensus establishing such ontologies, which are human artifacts, lead to endless committees meetings and large and fuzzy standards.

In many cases we need a mixture of domain-specific and general-purpose programming languages. Note that a DSL that is mechanically translatable to machine language is nothing more than a high-level programming language that does away with generality in favor of expressivity for the application domain.

As for programming languages, modeling languages come in many shapes and forms and UML is no more a Universal Modeling Language than Esperanto is a universal natural language. Modeling business interactions and machine flight are fundamentally different activities and are so at the modeling level.

Given the multi-language nature of large systems today, a language that plays well with others and recognizes the existence of other languages (DSL and otherwise) has a definite advantage. Unsurprisingly, the message of this section is that in addition to being a good play-mate with other programming languages, Ada needs to meld well with DSL for the domains Ada has been designed for: industrial systems. See http://www.open-do.org/projects/p/ for a possible approach in this area.

Raising the Level of Abstraction: Brick by Brick

There is a constant race to raise the level of abstraction. In the previous section we have looked at DSL as a possible way to raise the level of abstraction. Another way is brick by brick. If there are libraries, components, frameworks with the desired functional and extra-functional (safety, security …) behavior and properties that can be acquired cost-effectively we may as well use them. This will reduce our time to delivery and it will increase the quality of our apps while allowing us to keep the costs under control: the programming language here becomes the cement between the bricks.

Apart from things like standard libraries and containers, components are domain-specific. We are unlikely to find high-quality components covering a large spectrum of application domains that can all be used effortlessly in a single programming language. There is an interesting circular dependency (a bootstrap problem if you prefer) between the application-domain of a component and the language it is written in. We are back at the generality vs. universality dilemma.

In today's systems of systems with many connected devices, security issues are a growing concern. In addition to Ada's orientation towards safety, Ada could play the role of a glue language for certifiable/provable components with the desired security properties. Ada 4.0 has certainly made this possible. In this respect it is fascinating to go back to the "Finale" of John's 1983 book:

"Indeed, in the imagined future market for software components it is likely that packages of all sorts of generalities and performance will be available. We conclude by imagining a future conversation in our local software shop.

Customer:	*Could I have a look at the reader writer package you have in the window?*
Server:	*Certainly sir. Would you be interested in this robust version – proof against abort? Or we have this slick version for trusty callers. Just arrived this week.*
Customer:	*Well – it's for a cooperating system so the new one sounds good. How much is it?*
Server:	*It's 250 Eurodollars but as it's new there is a special offer with it – a free copy of this random number generator and 10% off your next certification.*
Customer:	*Great. It is validated?*
Server:	*All our products conform to the highest standards sir. The parameter mechanism conforms to ES98263 and it has the usual multitasking certificate.*
Customer:	*OK, I'll take it.*
Server:	*Will you take it as is or shall I instantiate it for you?*
Customer:	*As it is please. I prefer to do my own instantiation..*"

John and the Ada community had sensed the growing role that components were to play in the coming decades. Because large industrial systems is the domain where Ada made its debut and showed its strengths, significant sets of component libraries have not emerged from Ada to date. This state-of-affairs is part of the socio-economics of the domains Ada has targeted. This is very different from the status of the Java context where a large set of business-oriented components and frameworks have appeared: in the last two decades business apps have dominated the software and service industry.

Still what are we to do with the "certification" or "provable properties" aspects of the components John talks about in his fictional dialog? That is an interesting

alley where Ada 4.0 (with its assertions, pre/post conditions, and type invariants) should be leveraged on in the realm of provable/certifiable components in safety-critical and security-critical domains (for safety-critical domains DO-178C has created new opportunities for Ada 4.0 to lower the costs of certification). Efforts are ongoing in these areas at Kansas State University and other places such as in the Hi-Lite project (see http://www.open-do.org/projects/hi-lite/). These efforts have their foundations in the SPARK language and its vision.

The objective of Hi-Lite is to combine testing and formal methods to lower the cost of verification. The enabler is an "executable annotation language", which allows writing contracts on types and subprograms for unit testing (because it is executable) and unit proof (because it has a logic interpretation). Ada 4.0 comes with such an executable annotation language in the form of type invariants, pre and post-conditions for subprograms, and a rich expression language (if-expressions, case-expressions, quantified-expressions, expression-functions). Annotations can be written by the user, inferred by static analysis, or generated with the code from a model. Being able to apply formal verification to parts of a program and testing to the rest of the program will be key to lowering the costs of verification.

The message of this section is that Ada 4.0 (and beyond) could be used to create certifiable components (general-purpose and domain-specific: containers, TCP/IP stack …) for the domains Ada has been designed for: industrial systems.

Security

Security in mission-critical applications is a growing concern, and a difficult one. In a safety-critical system developers have to ensure that software malfunctions lead either to fail-safe modes or have to show sufficient due diligence so that chances of catastrophic failure are reasonably low (the famous ALARP – As Low As Reasonably Practical - principle). What these developers fight against is their own mistakes or unforeseen sequences of events in other software components or the natural environment with which the software interacts.

In a security-critical application developers are fighting against other humans of equal and sometimes superior intelligence that may try to exploit any breach in the software to take control of the underlying system. ALARP approaches are no longer sufficient, the challenge is much greater than in conventional safety-critical systems: we need to use formal approaches to demonstrate that the most critical applications are provably secure in the context of their use.

Multicores and Industrial Systems

CPUs have gone multi-core. Industrial systems are affected by this trend and will be even more so in this decade. With all their glory and glitter, today languages and their programming environments do not ease the task of writing concurrent applications to take advantage of multicores: it is both a matter of programming paradigm and tools. Ada is no different, except that Tucker Taft, the key architect of Ada 95 and beyond, has recently designed a programming language, ParaSail, to address the issue of programming multi-core (see http://parasail-programming-language.blogspot.com/). A subset of Ada 2012 is a very natural and sound basis on which to graft the concepts introduced in ParaSail for seamless programming of multi and many-cores.

Ada as a Pivot Language in Requirements-Based Development

An interesting role that Ada 4.0 can play in the context of safety-critical software is to facilitate collaboration and communication within a team and lower the cost for the production of certification artifacts. For more on this read the Ada Europe 2012 paper: “Source Code as Key Artifact in Requirements-Based Development: The Case of Ada 2012” by Comar, Ruiz, and Moy.

Tools and Programming Languages

Suppose I gave you the programming language of your dreams: The right level of expressiveness and efficient use of target hardware for your application domain, a clear and elegant syntax (textual or graphic). Need anything else? Well of course you do. Some 30 years ago a text editor and a compiler/interpreter were the only things you needed. Fast forward to 2012: try teach programming to young students providing just a text editor and a compiler, good luck! Leaving aside the importance of programming environments and libraries, if you wanted safety 30 years ago, the philosophy was to put the safety-nets in the language and compilers or run-time systems were tasked with spotting unwanted behaviors. Today there is an alternative. Use an un-safe or non-secure language and use advanced tools (e.g. based on static analysis) to detect unwanted behaviors in computer programs. Depending on the application domain this second approach makes sense.

On our travel from problem to computer-executable solution it does not matter how we got there. What matters is how easily we got there and, depending on the application domain, the quality of the end result. In this respect the environment that surrounds a given programming language matters very much, it is part of the “problem-to-solution travel experience”. As Erhard Plödereder said at the Ada Europe 2012 conference in Stockholm:

“Programming Language technology is increasingly “environmental” … the distinction [between] programming language and tool responsibilities blur”.

At the same conference José Maria Martinez Rodriguez added:

“When starting a new software development project you should take into account all the software development cycle and how a potential language fits in this cycle. As well as how the technology around this language helps or

assists in supporting the life cycle. For example, having the chance of generating code from design, a good and solid support for your language within your favorite design tool seems desirable. .... This "technology assistance" gets crucial in verification and validation since, in the context of complex systems, this task can be quite time consuming.

It is so important how language technology assists in the development of the final product, that sometimes it is easier to choose the language based on the surrounding technology ..."

Ada for Digital Natives

Programming language experts focus on language purity, elegance, and completeness. What do our young programmers care about? And who are these programmers anyway? Is computer science a specialist-only discipline or is it a skill that most scientists and engineers need to master much like the ability to speak English? Today there is a broadening and blurring of engineering roles. Engineers are required to have a hand in multiple areas. As a result a programming language for industrial systems should be attractive to engineers as well as computer scientists.

How will these generations of new scientists and engineers learn programming? This decade presents a fantastic opportunity: web and tablet technologies allow to easily reach current and future programmers of industrial systems. To be vibrant the Ada experience must be readily available to younger generations. These digital natives are tech-savvy, plugged-in, and require quick and convenient feedback.

The design complexity of modern programming languages, be they Ada, C++, or Java, is significant. John's 1983 Edition of "Programming in Ada" was 367 pages; John's "Programing in Ada 2005" is 828 pages. For C++ it is terrifying: the book on "The C++ Standard Library: A Tutorial and Reference (2nd Edition)" is 1128 pages, and that is just the standard library. Most programmers don't want to be gurus. We have to develop short, interactive, design-elegant, self-contained, on-line tutorials that present subsets of Ada in which useful programs can be written. Not just one tutorial. A family of tutorials depending on the concepts each tutorial wants to convey.

Regarding the approach to tutorials and teaching (on-line and off-line), I recommend the reading of "Programming goes back to School" in the May 2012 edition of the Communications of the ACM. The article promotes a "project-first" approach instead of the more traditional "principles-first" methodology. This pedagogical style allows students to learn principles just-in-time which proves to be very beneficial from the viewpoint of captivating the audience (and I believe speed of learning for many). The following diagram from the article is particularly telling. The explanation of the diagram quoted from Webb, Repenning, and Koh is fascinating. The quote is an excerpt of their article: "Toward an emergent theory of broadening participation in computer science education" published in the ACM Special Interest Group on Computer Science Education Conference in 2012 (SIGCSE 2012).

The fundamental idea of the Project-first approach can be illustrated through what we call the Zones of Proximal Flow (ZPF) […]. Flow is an ideal condition for learning […]. The ZPD can be understood as an orchestration of participation in a rich set of carefully designed practices where forms of assistance and tool use are strategically employed. In the Zones of Proximal Flow diagram in Figure 1, the horizontal axis represents students' computational thinking (CT) skills and the vertical axis represents the level of the design challenge that would be intrinsic to a certain game or STEM simulation [STEM = science, technology, engineering, and mathematics]. […] As student acquisition of skills advances in response to the challenges, an ideal path in the flow region would progress from the origin to the upper right. Within this diagram, pedagogical approaches can now be described as instructional trajectories connecting a skill/challenge starting point (A) with destination point (B) in the Zones of Proximal Flow diagram. In many traditional CS education models, a principles-first approach would introduce students to a number of concepts such as AI search algorithms that may, or may not, be relevant for future projects. At some later stage, students receive the challenge of making a project such as a Pacman-like game.

The acquisition of skills without the context of concrete challenges is not a bad pedagogical model, especially at the undergraduate CS level, but it runs the risk of seeming irrelevant, hence boring, for a broader audience of younger students if it does not go hand-in-hand with project based approaches. This assertion is consistent with the Flow model and with our own observations in classrooms. Instead of decoupling the acquisition of principles and the applications of these principles to a project, the project-first approach combines just-in-time CT skill acquisition with application to produce a tangible artifact.

In a nutshell: learn by doing, by example, by trial, by cut-paste-modify. Engage students in an exciting and feasible project. "I hear and I forget. I see and I remember. I do and I understand", Confucius.

In the end scientists and engineers want to build things. Can we craft tutorials where students experience the exhilarating feeling of building a system in Ada? Simulations on tablets? Lego Mindstorms, train, UAV, or robotics projects?

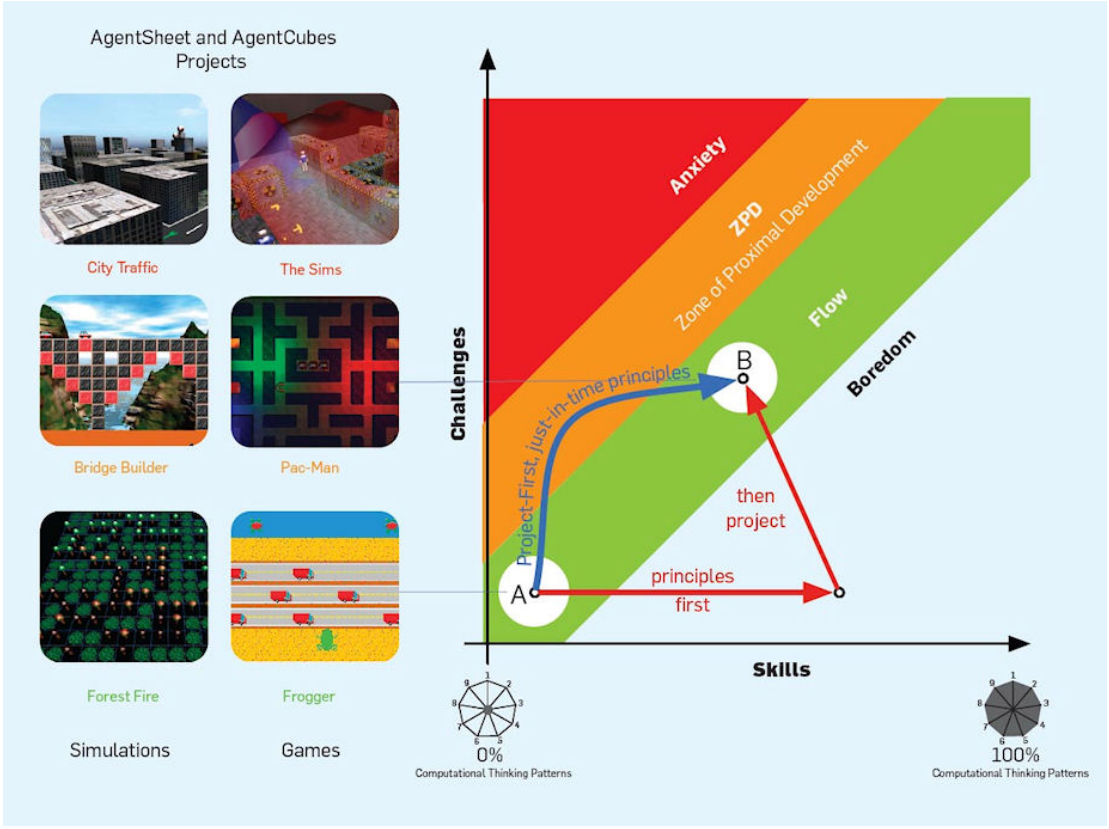

Figure 1: Zones of Proximal Flow

If we look at this issue from a different angle, it is important that we help universities increase their focus on software development for industrial systems. In these contexts the use of Ada, as one of the tools in the Swiss army knife of the engineer, is appealing and is to be encouraged by providing to educators ready-made chunks of self-contained and student-engaging Ada training material.

The message of this section is that we should teach Ada providing a level of immediate engineering feedback and gratification. For instance, why bug students with constraints that do not have a pedagogical purpose? Why don't we allow the direct execution of Ada programs asking the compiler to automatically fix silly mistakes like missing semicolons.

The entry point is the initial experience. If that experience is gratifying, if the student has learnt and built something by doing, Ada's usage in industrial systems will broaden as new generations of professionals enter the workforce.

Concluding Thoughts

Ada has been created for industrial systems embedded in an airplane, train, satellite, helicopter, UAV, subway, automobile, radar, medical device, … for industrial systems controlling air traffic, power plants, railways, … as well as simulators for all of the above. Ada has been very successful in these areas. Ada's strengths have shown that its use in other domains can bring significant advantages and rewards.

Today the software community is looking at the "Cloud". The fact that these new celestial systems have taken over the financial and socio-dynamics of the computing industry and use languages and technologies that are intertwined with the history, evolution, and rise of the Web is not contradictory with the strengths and use of Ada for the domains Ada has been designed for: earthly industrial systems where software matters.

To keep playing an important role in future industrial systems, Ada's level of abstraction in describing these systems, should continue to rise, while attracting new generations of users. For this to happen, Ada language designers and tool providers should continue their cross-fertilization journey towards model-based and formal methods approaches integrating multi and many-cores in the equation. The Ada community must develop exciting Ada tutorials and should help teachers develop engaging courses on software development for industrial systems.

Thank You

Many thanks to Ed Schonberg, Ben Brosgol, Yannick Moy, Nicolas Setton, Ed Falis, Greg Gicca, Tucker Taft, Eric Botcazou for their feedback on initial versions of this paper. Many thanks to Erhard Plödereder, Bertrand Meyer, and José María Martínez Rodríguez for very interesting discussions at the panel of the Ada Europe conference in Stockholm on June 12, 2012 . Special thanks to John Barnes who decade after decade has helped the Ada community with his prolific and thought-provoking writings on Ada and SPARK.

Panel discussion: What is language technology in our time?

Erhard Plödereder
University of Stuttgart, Germany

Whenever I am asked about the direction of language technology, I fetch my crystal balls, dust them off, and try to decipher the visions that they provide. Yes, plural, because I have two crystal balls. One has a rosy hue and it shows me the future as I wish it to be. What I see in it are languages that meet three needs: the creation of reliable software, as we increasingly entrust our life, money and privacy to computers, code that can be parallelized easily, as many-cores are the way of the future, and code that can be quickly produced and maintained, as this is a driving force in the producing industry. The first part spells "Ada" to me but only after a long stay in a weight-loss and rejuvenation clinic, or more likely one of its spiritual descendants. The middle part spells "functional" to me, but I take any model that tightly controls and minimizes mutable global state, and promises the absence of unintended race conditions in the code. And the last part spells "program generation" and "language environment". Finally, 30 years after some of us have started pushing the notion of supporting software production by environments that are substantially more encompassing than just editors, compilers and debuggers, we find a wider distribution of language environments in use out there. Major collections of reusable components are available, and tools help us in building the new software. The tools still are not as fancy and functional as we envisaged them a few decades ago, but certainly a far cry beyond command-line invocations of compilers, linkers and debuggers. As a consequence, the dividing line for responsibilities between programming languages and supporting tools will become more diffuse in the future. Checks that today we see as language-enforced might migrate into the realm of tools. The strictness of the language models oriented on the needs of separate compilation will be relaxed in a world where checks involve global analyses. In fact, I recently talked to a computer scientist, who believed that all individual compilations were based on global recompilations of the enclosing library already. And, indeed, the collective speed of our multi-core computers is close to making a global compilation indistinguishable from a strictly separated compilation of individual units. The use of specification languages and their associated program generators will continue to increase. Here, too, the border line between specification and programming languages will blur. On this last point of environmental embedding of languages, but alas only on this one, my second crystal ball agrees with its sister, although it cautions me to not expect it to happen as quickly as technology would actually allow.

My second crystal ball has a brownish hue from all the CO2 and other waste and detrius that reality unloads on us. It displays pictures that others might rejoice about while I find some of the produced waste too poisonous to join the happy fray. I see a steadily growing use of dynamicity in programming languages, notably very successful scripting languages that make it easy to mash applications, applets, apps, and 'a's, into amazingly complicated networks of interacting components, and all so quickly that us old fogies that insist on static checking of programs are left behind in the dust created by amazingly agile and extreme programming. Still, I am quite worried that my tires are being pressurized by "int" or "float" and not by "bar", "atm", or "psi". The lack of redundancy in specification languages causes static checks to become increasingly irrelevant, since there is very little left to check against. Consequently we have no verifiable notion of "correct" specification. Instead we need to execute or simulate the execution of the system by model checking to assess the plausibility that the specification or code is indeed the intended one. In as much as the languages are concerned, I wonder about the audacity of their designers to ignore two thousand years of unchallenged knowledge: "errare humanum est" or, as Alexander Pope extended it so aptly: "To err is human, to forgive divine." In modern computer science lingo, divine forgiveness is termed "duck typing", a principle that, in social sciences, is also known as "I can do no evil" or "because I say so, it is", which is a position usually reserved for deities, absolutistic royalties and dictators. Or is it that the designers wanted to create forgiving and hence divine languages? More seriously, there are of course reasons why flexibility and dynamicity of execution is needed, particularly as we try to make components cooperate that meet for the first time during the execution of complicated systems. My question is whether we should allow these principles to invade all other domains of software. My crystal ball immediately replies that the question is ill-posed: it is not a question anymore, it is reality already. These flexible languages are immensely popular and, by the traditional inertia of language usage, will still be there many years from now. The practical view therefore is that we have to deal with the unwanted consequences. And maybe there is gold at the end of this multi-hued rainbow after all. Our risks are not only the languages that endanger the reliability of our systems. It is also the growing complexity of the systems themselves that makes their behavior so fault-laden. Programming languages have only limited impact on improving this situation. Maybe it is time to give up the

goal of correct programs. Maybe future languages should put more focus on dealing with the malfunctioning of components and subsystems as a whole, be it from avoidable programming errors, or more deeply rooted design errors, or even ill-conceived requirements. Maybe we can employ some variant of swarm intelligence to recover from malfunctioning and massively faulty individual software components. After all, the human race has successfully applied this strategy in dealing with the fact that to err is human. Of course, one has to know that there is also something known as swarm stupidity, but, knowing so, we can try to prevent its bad consequences. I do, however, have definite problems with divine forgiveness as the answer.

Who will take the lead? Or, put differently, which language will succeed? I keep claiming, based on the evidence of the past, that there are only three reasons that can make a language successful. Technological superiority is not among them. The three reasons are: firstly, a major company must support the language by decisive marketing and financial investment to create the grass-root support for the language. Secondly, the language must be perceived as opening a new application domain – anybody not getting aboard, must be afraid that he will miss the boat. Here, marketing plays a major role. Alternatively, the language must appear to allow for a completely painless, i.e., fully upward-compatible, transition for users of an older language. Users will migrate to more modern alternatives if they begin to be malcontent with an old technology and can bring all their human and product assets with them into the new technology. Unfortunately, the programming language community failed to produce the "social network language (SNL)", be it as an upward extension of an existing language or as a new language that one couldn't do without. If pushed by a big player, it could not have missed to be a major success.

Discussion Panel: What is language technology in our time?

José María Martínez Rodríguez
Software Engineering Manager, CASSIDIAN

Abstract

This resume describes the position of the panelist regarding language technology in the context of embedded real-time software development for airbone systems. These considerations can be extended to those areas where high integrity systems are present.

Language Technology

Language technology covers all the facilities built around a specific language that help software-based systems development. The concept is wider than language specification itself (plus compiler, debugger, editor) and includes libraries, design patterns, frameworks, development tools, etc...

Systems are increasing in complexity by the time as well as the design, coding, verification and validation tasks needed to develop such systems. This trend has led to extend language technology definition.

When starting a new software development project you should take into account all the software development cycle and how a potential language fits in this cycle. As well as how the technology around this language helps or assist in supporting the life cycle. For example, having the chance of generating code from design, a good and solid support for your language within your favourite design tool seems desirable. If you need some graphics support, having a good and mature library suitable for your desired language is also preferred. This "technology assistance" gets crucial in verification and validation since, in the context of complex systems, this task can be quite time consuming.

It is so important how language technology assist on the development of the final product, that sometimes it is easier to choose the language based on the surrounding technology than in the advanced features of the language.

Ada language specification is growing with new features and "heavier" runtime whereas current language usage in high integrity systems sometimes is reduced and bounded, typically zero footprint or ravenscar-like profiles are being used. There is a necessity to strip unneeded features/functionalities. It is also remarkable to point out that nowadays it is not so rare to see Ada95 as the preferred language baseline for new developments. For example, the simple you keep it the easier/faster the verification can become.

Considering this scenario, Ada should adapt to current user usage providing more profiles depending on application domain and, somehow, more involvement on current tool ecosystem. Since "corporate tool" concept is always present on the industry (only one tool for all the company needs), market well known development tools should have a robust Ada support on their features making it suitable for a company-wide mixed languages environment (one tool - multiple languages).

Ada should keep in mind that choosing a language for a new software development is like buying a car: you may have chosen the car segment (language characteristics) but at the end of the day, the car's equipment (things that would made your "life" easier: tools, libraries, etc...) and total investment will decide the winner. And this implies not only a question of what language you choose for a project, it also helps in creating a "language culture" within a specific company.

Language technology is so important that can be a mean to catch users for a language. Sometimes it is the technology that leads you to choose a language and not the other way round (as one may think as the proper way).

Ada community has and important challenge in building and consolidating a state of the art language technology that could attract new users and keep current ones.

Rules for effective language design

Bertrand Meyer
ETH Zurich, ITMO (Saint Petersburg) and Eiffel Software

Eiffel is more than a language and is best characterized as a method of software development, addressing the full software lifecycle. The language exists to support the method. Both its original design and its evolution over 26 years have stuck to a number of principles [1], including:

- "One good way to do anything": make sure the constructs of the language are powerful and expressive, but do not require language users to choose between alternative ways of achieving the results (such as, in C++, calling a dynamically defined function through indexing in an array of function pointers, or using O-O-style dynamic binding).
- Throughout the language design and the resulting style of software design, apply thoroughly and consistently the ideas of object-oriented development, meaning abstract data types.
- Obviously, Design by Contract, as a basis for correctness but also for exception handling, a proper treatment of inheritance, built-on documentation, built-in verification.
- Keep the language consistent and simple, by making sure for example that the numbers of keywords (about 65) and constructs remain manageable.
- Define strong style guidelines along with the language definition proper; for example, every routine is expected to have a header comment explaining its purpose, with a standard style for how such comments should look like, and hooks for them in compilers, documentation tools and other parts of the IDE.
- Focus on helping programmers write correct, robust programs, through techniques such as strong typing and the more recent "void safety" mechanism which guarantees the impossibility of null-pointer dereferencing.
- Acceptance of the inevitability of evolution, as new ideas emerge (such as agents, influenced by mechanisms from functional languages and providing higher-level functionals within an O-O context) but with the constant constraint of keeping the language simple.
- As a consequence, acceptance that language mechanisms may be removed, provided a transition path is available to language users, who are given time and tools to upgrade to the new, better facilities.

Ada was very much the reference when Eiffel was first designed; Eiffel is a very different language, based on object-oriented concepts, but the comb-like keyword-based syntax resembles that of Ada, with simplifications — for example, end is just end and not qualified. [1], published in 1999, explicitly mentions Ada, on the topic of how small a language should be:

> *We could paraphrase a famous quote and state that a language should be as small as possible but no smaller. That doesn't help much. More interesting is the answer Jean Ichbiah gave to the journalist who, at the time of Ada's original publication, asked him what he had to say to those who criticized the language as too big and complex: "Small languages", he retorted, "solve small problems".*
>
> *This comment is relevant because Ada, although undoubtedly a "big language", differs from others in that category by clearly showing (even to its critics) that it was designed and has little gratuitous featurism. As with other serious languages, the whole design is driven by a few powerful ideas, and every feature has a rational justification. You may disagree with some of these ideas, contest some of the justifications, and dislike some of the features, but it would be unfair to deny the consistency of the edifice. Consistency is indeed the key here: size, however defined, is a measure, but consistency is the goal.*

It has been a challenge, but also an ever exciting endeavor, to grow the language over the past three decades while maintaining that consistency.

Reference

[1] B. Meyer (2000), *Principles of Language Design and Evolution, in Millenial Perspectives in Computer Science*, Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, eds. Jim Davies, Bill Roscoe and Jim Woodcok, Cornerstones of Computing, Palgrave, Basingstoke-New York, pages 229-246, correct text at http://se.ethz.ch/~meyer/publications/hoare/evolution.pdf

What is Language Technology in Our Time?

A Summary of the Panel Session of Tuesday June 12th

Albert Llemosí
Universitat de les Illes Balears, Spain

At the start, Tullio Vardanega in his role of session chair stated very clearly the rationale for choosing language technology as a subject for a panel: a language is a view of the world and the aim of language technology is to provide the appropriate expressive power for such a view. Indeed, these words are an obvious rhetoric overstatement but they are quite precise as long as we take "a view of the world" as a synonym for "an approach to programming practice and style". What can be expressed in the language exists, what cannot be expressed in the language does not, and therefore, the aim of language technology must not be a matter of syntactic details but a matter to provide the programmers with the exact expressive means that match their needs in the most precise way.

The first panel speaker was Bertrand Meyer, a very well known researcher as the designer of the OO language Eiffel and the architect of its related software. He had been the keynote speaker of the same day, so his words in the panel were unavoidably related with the longer exposition he had done a few hours ago, and therefore a short comment about that keynote should be mentioned here, even though the scope of this small report is constrained to the panel session. The title of Meyer's keynote talk was "Life with Contracts". The topic could have offered him the opportunity to say that contracts were invented in Eiffel and that he was glad to see them now considered in Ada 2012 and to compare both approaches. Instead, he focused his speech mainly on syntactic details. Meyer's speech in the panel session went basically the same way. First, he outlined the fact that much has been said about language design, but very little about language evolution, which is an essential aspect of programming languages. Then he balanced the alternatives of extensive democracy and restricted meritocracy in the design of the language pointing out that the right approach is to be dogmatic when it counts and flexible when you want to help the users rather than standing your way.

The speech included a eulogy of how carefully Ada was designed and evolved; making particular emphasis on the existence of a Rationale that shows that every feature has its justification. Even if one disagrees with some of the design decisions, nobody can deny the consistency of the language. He also stated that both Eiffel and Ada are designed to provide support to software engineering principles as much completely and coherently as possible.

The next speaker was Franco Gasperoni, from AdaCore. He made a great eulogy of the principles that guided the design of Ada and the way they were exposed in the famous book of John Barnes, *Programming in Ada*, with particular emphasis on the second edition of the book, which corresponds to the 1983 version of the language. He also cited some of the comments that appeared in the introduction to that book, which were written by Jean Ichbiah.

Gasparoni remarked that we cannot expect that a programming language may become absolutely universal, because the requirements in different domains are so different that no language can cover all them. Therefore, we should not consider that Ada is not a success only because it is not used in all domains. Ada was particularly designed for industrial systems, and in this domain it has been successful and it is still in use.

Gasperoni's speech concluded with some remarks concerning the teaching of programming. The way the new generations become educated in programming is essential both for the quality of the software and for the success of Ada. He presented a diagram to illustrate two possible approaches for teaching programming. One path started with a description of programming principles and language features and then turned into the increasing complexity of the systems addressed, and the second path started addressing complex systems, even with a small set of language features and very elementary programming principles, so that the extension the repertoire of features and techniques can be seen as a means to provide a better support for the difficulties found during the construction of such complex systems. He clearly advocated the second approach because he considered that the first one is very boring for the students.

The third speaker was Erhard Plödereder, from the University of Stuttgart, which provided a more academic view. He started pointing a few facts about the current state of programming practice, some of which were more desirable than others. The first fact is that programming technology is increasingly environmental, i.e. it relies not only on compilers and debuggers but on much larger toolsets and, moreover, it is becoming a compositional activity based in combining external components that we are not responsible for. The second fact, to some extent a consequence of the first one, is that software construction is increasingly experimental. Most programmers don't start from thinking, but from seeing what happens.

Luckily, there are a few niches where program understanding and analysis is still essential. The third fact is that programming language tendencies stretch between two extremes: some of them try to express the nature of the problem regardless of its possible implementation, whereas some others precisely focus on how the problem is effectively mapped onto the target machine. As the OO approaches are in the first set, they are probably ill suited to deal with parallel processing in multicore architectures, which is, no doubt, the future of the hardware platforms.

After having provided this academic view, professor Plödereder turned into the reality view, and stated that the technological superiority is not a guarantee for the success of a language. Instead, the main factors for the success are the support of a major player, such as Microsoft or Oracle-Sun, the appearance of a new application area that is deemed unreachable without the new language (most likely, cloud computing and social networks shall require the development of new languages) or a fully upward compatible improvement of a successful but aged language.

The last speaker was José-María Martínez, a software engineering manager at Cassidian, a part of the EADS consortium that is devoted to airbone systems. His view was, obviously, that of a developer of high-integrity real-time embedded software. He stated that the programming language is only the lowest part of their V-shaped software life cycle, which includes a big set of standards to follow, including procedures, documents and the certification process, with many people involved.

Regarding the selection of the language, the decision is influenced by many factors, and technological superiority is not the major among them. Instead, the major factors are the task to be performed, the certification regulations to fulfil, the company standards, the schedule, the budget and the past, i.e. the languages the people involved are familiar with, the work done that can be reused, and the available tools.

After these short presentations by the panellists, a discussion started with the participation of the audience. Ian Broster said that nobody had mentioned how a language can be made “cool”. By cool, he meant what the young people seem to know about. The answer from Erhard Plödereder was that cool stuff gets hot very rapidly, and later cools again, and that the problem of language technology is that it takes a very long time to mature, to be produced and to be targeted to a particular area, which will be a difficulty for the future generation of programming languages. José-María Martínez claimed that Ada is hidden, that there is no advertising for it and that a Web page should exist for this purpose. Franco Gasperoni said that cool means fashionable and that fashion comes and goes. He remarked the position of Erhard Plödereder's panel that a programming language is cool because it is associated to a cool set of applications that you couldn't do before, and Java is a clear example of this. Then he suggested that there should be a sociological study about the human aspects of coolness in programming languages.

Ben Brosgol pointed that two things seemed contradictory in Erhard Plódereder's presentation: that OO is dead and that the future is C# scripting. Erhard's reply was that there two sides to his argument: the first part was the academic view and the second part was the reality view. He insisted on the fact that technologies are successful not because they are technologically superior. Obviously, it helps, if they are, but it is not a decisive factor. His particular opinion is that the future is functional simply because the distribution of functional software is so much easier than that of OO software. Then Bertrand Meyer commented to oppose to this latter statement. He told that he had recently published a paper comparing functional and OO languages and that the problem of functional languages is that they don't scale up. Instead, OO design and OO technology are there to address the real problems of software engineering: they provide clear guidelines about how to build, grow, maintain and evolve large software systems, taking "large" in the sense of number of people involved, the time they must remain in existence and the number of changes they have to handle. Functional languages have some very nice ideas and they are very elegant, but in terms of modularity they don't provide anything. Meyer wanted to convey that he wasn't saying that functional programming ideas were not useful: Eiffel and C#, which are very successful in scaling systems up, have also been successful in integrating some functional programming principles. However, for designing software architectures, which is what software engineering is about, OO decomposition is the useful approach, which in turn is the application of abstract data types, which in turn is the application of the scientific principles of separation of concerns and abstraction.

Franco Gasperoni asked the audience about how many among them felt themselves as computer scientists and how many as engineers. The interesting thing is that some people raised their hands twice, which allowed Franco to set the question of the melting of (or the fighting between) these two opposing views in the working environments of the people attending the panel. The most common view was that people working in industrial environments see themselves as engineers (software, mechanical, control or whatsoever) and that, in principle, as engineers they do not aim to improve but to put to use the sciences their technologies are based upon. The problem with software engineering is that its link with its scientific grounds is very weak. No other engineering domains (mechanical, chemical, ...) can achieve any useful result without a deep knowledge of the physical sciences they make use of. Erhard Plödereder noted that there is a significant difference between software and other engineering disciplines: a civil or mechanical engineer is responsible of the system he designs because the problems that he solves are in the scope of what he is able to control and predict. Instead, for a software engineer there is no way of predicting the behaviour of a system but to put the system into operation. The computer science fault is that it does

not provide the appropriate science. It has mathematics in its grounds but not those that the software engineer would need.

Bertrand Meyer answered that the situation that Erhard had described is deplorable but that he was sure that it will change soon. According to him, it would be inconceivable that if a car doesn't work, the excuse is that the mechanical engineer hadn't wanted to learn differential equations. Likewise, it should not be acceptable – and does not make sense – that we refuse to teach simple things like contracts to the software engineers just because the programmers won't make use of them. However, this is not going to last forever and it is changing quickly. To be more precise, we have to distinguish among three areas of the IT industry because if we mix all them together we are never going to get a good analysis. The three areas are casual programming, enterprise programming and mission-critical programming. Casual programming is what everybody does, including people with no computer science training at all (Web sites, Java scripts, ...). In contrast, in the area of mission-critical software, this kind of discussion does not exist anymore: the people in that community have accepted that not to use the appropriate methods means killing people. To Meyer, the interesting part is the enterprise area, which involves very important financial and commercial applications whose misbehaviour may have very important consequences, but not visible enough because they prevantely entail loss of money but not of lives. So, they haven't yet realized how important it is to move to an engineering mode of working, but this will happen. So, he is sure that we will not have the same kind of discussion in five or fifteen years from now.

Then Marco Panunzio expressed that he was happy that modelling had been mentioned in some of the presentations because he is convinced that modelling is becoming increasingly important, as a lot of things can be done there, such as verification of properties and code generation. Apparently, it might seem that, as a consequence, the programming language is becoming less important, but he thought that this is a more debatable conclusion. Therefore, he wants to know the opinion of the panellists about if (and in what area) the programming language matters and the things that a programming language should offer to complement what is done at the modelling level.

The answer from Erhard Plödereder was twofold. First, that it should be clear that the language that describes a model is also a programming language, especially if it is generative. Second, that the redundancy disappears with the modelling approach: there is nothing one can validate against, just because the model is always correct. He considers this extremely dangerous. Then Bertrand Meyer objected that the revolution about OO technology has been to unify concepts of software development all the way from requirements modelling and so on, and this has made the success of OO programming. He added that now we have seen attempts to reintroduce distinctions that OO programming had removed, such as the distinction between models and realizations of the models, which is the biggest step backwards in the history of software engineering ever, because it turns it into a monster of complexity that leads to never ever model. He very much insisted that the modelling powers of OO languages are such that they can accompany us all the way from the most abstract study at the non-imperative stages down to the implementation and go back and forth. And, according to Meyer, this latter ability is essential because the problem of the software is not only building a system but also making it to evolve through the years.

Then José-María Martínez explained briefly some experience in his company that was in favour of Erhard's position. A couple of years ago they started thinking about using a model-driven development in a project due to the verification facilities of the modelling language. However, there was no way to control if the timing requirements were fulfilled during the modelling process. Franco Gasperoni added that modelling is a too general concept and that we should use the term domain-specific languages. For certain things, domain-specific languages are very interesting (for instance, writing automata) and that Ada has an important role to play in industrial contexts, possibly melting well with other domain-specific languages.

At this point, Tullio Vardanega, as the chair of the session proposed to the panellists to propose a final set of recommendations, especially what would they recommend language technology of the future to be.

Bertrand Meyer's started expressing how exciting it has been to participate in the design of a language and its evolution along 25 years as an inter pares, not necessarily primus. He obviously agrees with Franco when he says that no perfect language exists. However he considered that the requirement of having a different language for each particular domain (for instance, graphic interfaces or mobile phones) should be admitted as a failure. A programming language should provide good structuring mechanisms, and domain-specificities should be relegated to the libraries. He also remembered a comment from John McCarthy that languages are not designed but discovered, because a certain view of the programming activity pre-exists and the programming language raises as the expressive means that best matches this view. Therefore, his conclusion was that, given a certain view of the software, it is possible to get as close as humanly possible to the asymptote of perfection to design a language that should eventually cover in the end the entire application domains.

Franco Gasperoni decided to conclude by taking some pieces of sentences said by various people in the session, both speakers and the audience, and putting them together. He addressed first the question of the focusing of languages upon users (as representative of target application domains); three clear domains have been put off (casual, enterprise and mission-critical), and it is important to decide to whom we talk about, as they have

to be dealt with separately. The second important matter is that the difference between programming languages and tool responsibilities is blurred. One can do nothing with a programming language unless there is a complete set of tools available surrounding it. The third conclusion is that software engineering and computer science are working and need to work together more closely. The last important point is that teaching is fundamental for the success of a programming language technology. This is to be done by the writing of cool applications that are part of the target domain that we want to get.

Erhard Plödereder started expressing his agreement with Franco's last point. In terms of answering the question of what is the ideal technology, his view is a series of languages that at every level gives him the highest expressiveness for his purposes. Namely, an architecture language that allows him to design subsystems and systems regardless of any paradigm at all, a language for defining the models and simulate how the system is going to behave, a language for expressing contracts, both at the model level and at the implementation level, and so on. To sum up, the important thing is to have a language at each stage of a project that allows one to express the important aspects of that stage.

José-María Martínez also started expressing his agreement with Franco's points. He stressed that a language is useless if it is not complemented by a complete set of surrounding tools that provide a proper benefit by themselves. Moreover, a language technology must be mature, otherwise it is not sufficiently sound and robust to be applied at production levels.

The session concluded with a sonorous applause to the panellists and the chair.

Reliable Software, a Perspective from Industry

Jørgen Bundgaard
Ada in Denmark

Abstract

The invited panelists discussed what they see as the most pressing and challenging industrial needs in the way of software technology to facilitate the production of reliable software, such as:

- *Quality and safety standards*
- *Life-cycle models*
- *Processes, methods, techniques*
- *Languages and tools*

Questions to the panel:

- *If we have the right people on the team what else do we need, really?*
- *If we had a very pressing need, we would apply our engineering skills and solve it, wouldn't we?*
- *Are the "traditional" quality and safety standards a burden or a help?*
- *What do you think will be the next "Killer app" in software engineering?*
- *What are the main obstacles to innovation in our industry?*
- *When will we ever learn to estimate (development cost) correctly?*

Position of the moderator

I have always felt that the "most challenging need" - when starting a software project – is to have a development method with processes that work well for the given case.

For a software product in the maintenance phase this is usually not a problem. The work typically consists of planning, change management, changing the code, verification, regression- and validation testing, release management, and documentation. It is not too difficult to establish and optimize processes for that, and tools are usually available from the development phase.

The key is that the team gets to execute the processes multiple times, with opportunities to fine-tune the steps in the processes.

For a development project similar to one the team has done before there is at least the opportunity to look at the history and decide if changes are needed to the development method and processes used previously.

The key is to have people on board who have "done it before" and who can adapt the "standard" development method and processes for the specific project at hand, including how to comply with applicable standards, like DO-178B (aerospace), ISO 26262 (automotive), EN 50128 (railway), ECSS-E-ST-40C (space), etc. Organizations at a certain maturity level have "documented" and "repeatable" processes.

For a development project, which has aspects very different to anything the team has done before, the method and process issues are more problematic. It may be a project involving a new type of stakeholder, a project on a much larger scale, a new domain, a new standard, a new architecture, a new technology, a new role, and so on. Established processes may be inadequate to cope efficiently and effectively with the new challenges, risks, and opportunities.

The key is to get people on board who have "done it before". They may come from outside the team. For example, external "consultants" can, on the basis of experience, demonstrate the need for extra rounds in requirements clarification with the end users. They can show how to fulfill requirements like "The planning of the software integration shall describe the steps for integrating the individual software units hierarchically into software components until the embedded software is fully integrated" (ISO 26262-6). Or they can propose tools for demonstrating 100% MC/DC in unit testing, and much more.

Alternatively, the team members will have to figure it out by themselves, and if so they will do wise in clearing their approach in advance with auditors and assessors, if the project outcome is subject to formal approval.

There is evidence for the need of ongoing process definition and adaptation. Large established organizations use consultancy services to help defining processes for model based software engineering, to develop guidelines to comply with functional safety standards, and to advance from one SPICE level to the next.

From a software project management point of view it makes very good sense to ensure that (most of) the necessary steps in the development process have been identified, understood, and validated (if possible) in advance. The risk of unpleasant surprises (of many kinds) will be reduced.

On the other hand, there is a justified fear among managers as well as engineers that mandating compliance with a comprehensive process and standards library will

prevent the use of common sense and will be counterproductive.

The key is to apply a balanced view where common sense is in the high seat and the involvement of the people who must live the processes is mandatory. After all, it seems like good idea to work on a project where we have faith in our approach! NB! This should not stand in the way of innovation. Promising new ideas should be applied to important, but not critical work.

In conclusion, "the most pressing and challenging industrial needs in the way of software technology to facilitate the production of reliable software" are to have people onboard who have done it before – plus development models, processes, and checklists – made by people who have done it before – but are open to new ideas!.

Reliable Software, a Perspective from Industry

Ana Rodríguez
GMV Spain

In the last decades, the lessons learned coming from many critical software development processes on the different domains have led to the definition of a number of standards compiling good practices, recommendations or even mandatory rules to be applied in software development and, very specially, in software verification processes.

Dependability and safety are issues of paramount importance in the development and operations of critical systems. The contribution of software to system dependability and safety is a key factor, especially in view of the growing complexity of the software used in critical applications, together with the increasing cost and schedule constraints. For example, the need for more dependable and safe software has led to ESA the publication of "Software Dependability and Safety Handbook" (ECSS-Q-HB-80-03C) meant to provide guidelines on the implementation of the software dependability and safety requirements and on the application of methods and techniques for software dependability and safety.

Failures are caused by different reasons. Here are some possible ones:

- Software requirements are incorrect, incomplete or ambiguous. They can include unhandled states or unhandled environmental conditions, non-conformance in the software or deficiencies in the code (they cause software faults).
- Software requirements have not been implemented, validated and verified properly.
- Software has not been tested enough or has been tested inadequately.
- Software Defects.
- Software is used incorrectly.
- Poor design or implementation.
- Rare events can lead to uncontrolled states.

The consequences of a failure vary but may be severe. On the one hand, failures due to inadequate designs or implementations are easier to detect and solve. During validation and verification phases, it is checked that everything is in conformance to requirements. So, wrong designs and implementations are discovered.

On the other hand, unexpected behaviours or states due to different failures are more difficult to detect and therefore to solve. But the entire software must be designed in such a way that any single or combination of failures does not cause both critical and catastrophic consequences. Generally, a Safety Critical System must include proper mechanisms or means to guarantee dependability and safety at any level:

- Software fault prevention: Avoidance and/or reduction of fault causes before software go operational.
- Software fault tolerance: Ability of a functional unit to continue running correctly despite the presence of faults or errors.
- Software fault removal: To identify and remove the presence (number, seriousness) of faults after they occur. Fault removal takes place either during the development process or after system goes operational.
- Software fault forecasting: Software fault forecasting is used to predict software behaviour when a fault occurs. This involves how to estimate the present number, the future incidence and the consequences of faults.

To begin with, to make software as fault tolerant as possible, the Single Points of Failure (SPF), a part of a system which, if it fails, will stop the entire system from working, shall be minimised. The implementing fault tolerant measures can be done at the following levels:

- System-level measures: Fault tolerant measures such as redundant networking equipments, redundant storage, a carefully planned operational and maintenance strategy, and a carefully planned monitoring strategy.
- Component-level measures: At building block level, you should use fault tolerant components such as power control, Error Correction Code (ECC) memory, and redundant fans. At the network-level, you should implement fault tolerant networking components such as redundant switches, routing, and wiring.
- Operating Systems and SW environment measures: Before implementing specific component-level and system-level fault tolerant measures, there are certain operating systems, Software Development and Operational Environment measures to consider.

There are many fault tolerance practices currently applied in the different technology domains and software failure propagation prevention is actually taken into account in the Standards. The maturity, qualification status, availability and portability of the different methods and tools need further research.

Reliable Software, a Perspective from Space

Steen Palm

Terma A/S, Vasekær 12, 2730 Herlev, Denmark; Tel: +45 4594 9665; email: sup@terma.com

Reliable software is a must within the space community. A small software bug can result in the loss of a satellite, which has been built over many years and whose development costs are measured in hundreds of million euros. Therefore, the European Space Agency (ESA) has very strict requirements to the software development process and the quality of the final software. These requirements are documented in a set of so-called ECSS (European Cooperation for Space Standardization) standards that must be followed on every ESA software development project. The ECCS standards very precisely define activities to be performed, designs to be constructed, documents to be produced, reviews to be held, tests to be performed, etc. The overall objectives of the ECCS standards are to ensure reliable software products that will not fail during operation.

However, to follow the ECCS standards requires a substantial effort, especially because it involves an overwhelming number of tests at many different levels ranging from unit testing of software modules to assembly and integration tests on different satellite models. At the same time, satellites become more complex, the on-board software has to perform more advanced tasks, and the time available for the development decreases, while the requirements to the reliability of the final software product are unchanged.

ESA is aware of this problem and has taken an initiative to improve the way that the European Space community builds avionics subsystems. This initiative is called SAVOIR (Space Avionics Open Interface aRchitecture) and has taken inspiration from AUTOSAR (AUTomotive Open System ARchitecture), although the underlying industrial business model is different. The space community is smaller, the production is based on a few spacecrafts per year, and there are industrial policy constraints. Still, there is a need to streamline the production of avionics software and improve competitiveness of European industry. Reference architectures, reference specifications and standard interfaces between building blocks are an efficient mean to achieve the goal. Reusing specifications is expected to allow reusing products.

A promising aspect of SAVOIR is that it builds on the outcome of the ASSERT (Automated proof-based System and Software Engineering for Real-Time systems) project. The ASSERT process aims to enhance the system & software engineering activities by means of model-driven and property-preserving methods and automation tools. A main feature of the ASSERT process is that the software design is Ravenscar compliant by construction, implying for instance that timing properties of the system can be analysed by means of a schedulability analysis.

At present, a SAVOIR software reference architecture has been defined and is being validated within a number of ESA technology development projects. EDISoft RTEMS, which is a real-time operating system, is an example of a reusable building block that fits the reference architecture.

The SAVOIR software development process is intended to make use of standard building blocks and to be supported by a tool chain. Tools supporting the SAVOIR process are currently under development based on a combination of Obeo Designer and TASTE (The Assert Set of Tools for Engineering). Until tool support is available, the tool set TASTE supporting the ASSERT process can be used to achieve many of the advantages of using SAVOIR.

Terma is currently using the principles of SAVOIR and ASSERT for the development of software for controlling two instruments that are to be mounted on the outside of the International Space Station. The instrument MMIA (Modular Multispectral Imaging Array) is intended to study the high-altitude electrical discharges in the stratosphere and mesosphere above severe thunderstorms, the so-called red sprites, blue jets, and elves, while the instrument MXGS (Modular X and Gamma ray Sensor) will observe terrestrial gamma flashes occurring during the severe thunderstorms.

Developing Reliable Software is Impossible!

Ricky E. Sward, PhD.

The MITRE Corporation, Colorado Springs, CO, USA.

As software systems become more complex, ensuring that operational systems are 100% free of errors is an impossible task. Complex software systems may include millions of lines of code, developed by hundreds of software engineers, and will include errors found only during operational use of the system. In 2002, NIST reported that errors in software cost the US economy $59.2 billion annually [1]. One approach to reducing software errors is to increase the testing, validation and verification processes required for certification of systems. These processes are often mandated via policies and procedures. Another approach is to increase the use of Formal Methods in the development of complex software systems and prove certain aspects of the software's quality. Using Formal Methods during software engineering improves the requirements specifications, reduces the introduction of errors, improves error detection, reduces overall cost [2] and can help us accomplish this impossible task.

As an example of high integrity, safety critical systems, we'll consider some of the systems being developed as part of Unmanned Aircraft Systems (UAS). Although the FAA has not yet mandated certification of UAS, the components may include safety-critical sub-systems such as flight control systems, sense and avoid and navigation systems [3]. UAS have been used extensively in defense operations, but there are hundreds of civilian applications for UAS that are emerging. The software systems used to control these UAS must be reliable and predictable as we move to a world where unmanned aircraft are flying in the same airspace as manned aircraft. A major question for UAS is how do we build trust in the software and systems that control these aircraft?

Currently in the US, unmanned aircraft are not permitted to fly outside of Restricted Airspace unless they have obtained a Certificate of Authorization (COA) from the FAA [4]. The FAA has begun to streamline the COA application process and increase the number of UAS allowed to operate in the NAS. For manned aircraft systems, the FAA approves all aerospace software-based systems using the DO-178C standard. DO-178C includes activities needed when using Formal Methods in place of conventional review, analysis and testing of software systems [5].

Some UAS systems have already started the process of adhering to the DO-178C standard. Because of this, manufacturers of emerging UAS systems may include Formal Methods as part of their software development process, which will reduce their need for lengthy verification and validation procedures. If UAS manufacturers use Formal Methods, they will need processes and tools that allow them to quickly deliver capabilities to their operational users. The current UAS market if very competitive and getting a validated capability to the field is the key objective of UAS companies. They need ways to quickly train their software engineers on software development using Formal Methods. They also need tools that assist the engineers with proof obligations and formal requirements specification.

Another emerging trend in UAS development is to build Command and Control (C2) systems that include the man "on" the loop versus the man "in" the loop. More automated functions are being developed to relieve the UAS operator of mundane, tedious tasks. As these systems mature, they will also be scrutinized for their reliability and level of trust in the system. C2 systems are particularly susceptible to cyber-attacks, which may include an adversary taking over control of the system.

To validate the security of UAS systems and classified systems in general, there has been an increase in the security testing and certification required for these systems. As the requirements for certification increase, one beneficial practice is to incorporate the fulfillment of these requirements as part of the software development process. This is in contrast to attempting to fulfill the requirements after the software has already been developed. In the Tokeneer project [6], Formal Methods were used to achieve a higher assurance level of the Common Criteria.

If software development firms are to build reliable software and pass these stringent certification requirements, they should build the fulfillment of the requirements into their processes. If they use Formal Methods to attain higher assurance levels, they should also be provided tools and processes that help to incorporate Formal Methods into their software development processes.

The task of building reliable software, free of error when delivered to operational users, may indeed be impossible. Formal Methods can reduce the testing, validation, verification and certification requirements of software system. Building Formal Methods into the software development process may help software systems achieve higher assurance levels. Providing improved Formal Methods tools and processes to software developers may help achieve this impossible task.

References

[1] G. Tassey (2002), *The Economic Impacts of Inadequate Infrastructure for Software Testing*, Final Report, Prepared for National Institute of Standards and Technology by RTI.

[2] D. Mery and N. K. Singh (2010), *Trustable Formal Specification for Software Certification*, T. Margaria

and B. Steffen (Eds.): ISoLA 2010, Part II, LNCS 6416, Springer-Verlag, pp. 312–326.

[3] *Safety-Critical Services for UAS*, retrieved from http://www.certon.com/uas-uav.php on May 31, 2012.

[4] *FAA Unmanned Aircraft Systems* (UAS). Retrieved from http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/systemops/aaim/organizations/uas/ on May 26, 2012.

[5] D. Brown, H. Delseny, K. Hayhurst, V. Wiels (2010), *Guidance for Using Formal Methods in a Certification Context*, ERTS 2010 – Toulouse, France.

[6] Tokeneer Project. Retrieved from http://www.adacore.com/sparkpro/tokeneer on May 27, 2012.

Reliable Software, a Perspective from Industry

A Summary of the Panel Session of Wednesday June 13th

José Ruiz
AdaCore, France

What are the most pressing and challenging industrial needs in the way of software technology to facilitate the production of reliable software?

Introduction

To start the discussion, Jørgen Bundgaard, from Rovsing A/S, stated that the current situation in the high-integrity industry is that there are many different and diverse safety standards, many different lifecycles in use, a variety of programming languages, and a huge variety of tools supporting the previous elements. This complexity needs to be addressed to produce reliable software.

Position statements

Ana Rodríguez, from GMV, believes that we are able to do software which is 99% reliable, with the help of existing guidelines, methods and techniques for building safe and dependable systems. However, things are made more difficult each day by the continuous increase in software complexity together with the increasing cost and schedule constraints. If we look at the origin of failures, there is a wide range of errors which may come from software requirements, design and implementation, validation and verification, or incorrect use of the software. The failures which are the most difficult to detect and fix are those caused by unexpected events. And then, We need to apply fault tolerant measures to avoid failures to have critical or catastrophic consequences. Hence, reliable software is achieved by accepting the existence of failures, and designing systems minimizing single point of failures applying fault tolerance techniques.

Steen Palm, from Terma, indicated that in the space industry problems may cause the loss of an expensive satellite, so on-board software must be reliable, and ESA standards have been developed to help having reliable products. He agreed with Ana Rodríguez in the trend to have more complex software which need to be developed more quickly and cheaper. One of the initiative which may help producing reliable software under such conditions is the use of component-based reference architectures, such as those being defined in the SAVOIR and SAVOIRE FAIRE projects. Another way to achieve reliability is by imposing a way to use existing reliability standards, because different groups of people use them in different ways with heterogeneous results in terms of reliability.

Ricky Sward, from the MITRE Corporation, believes that given the huge size and complexity of modern software, which is developed by many different people, there will always be errors. However, we are able to produce successful operational software with errors. Exhaustive testing and strict methodologies help producing reliable software, but still problems are found in the operational environment. He suggested that formal methods is the way to go as the most effective way to reduce verification and validation costs, increasing assurance levels, and helping detect problems early. It is important to note that the use of formal methods will require good training for the development teams, and tools and processes to incorporate formal methods into software development.

Discussion

After these position statements, there were questions from the audience. The first of them was about the need of good engineers to have successful projects, and whether the use of well-defined methods and advanced tools was there to try to fix the need of having well-trained people. The three panelist agreed on the fact that you need good people, but tools and methods help a lot in the different phases of software production, and they help reducing the cost and time-to-market. For example, correctness-by-construction approaches with formal methods are very useful for achieving reliable software, and they help reducing the cost of verification and validation.

Another issue that was raised from the audience is the terms to be used for software that needs to be correct. Reliable is probably not the best term (it implies probability analysis which is not probably what we want to apply to software). Safety is used in avionics, and dependability is also another term which applies well to software.

One of the problems to address in high-integrity systems is the need to think about all possibilities, and to have barriers for any failure. Fault tolerance techniques and exhaustive testing help a lot to achieve this goal, but improving the software development process may be more effective. In this line, requirements engineering was identified as one of the weakest parts in software development, which is often neglected. A clear definition of what needs to be done is essential to have a good understanding of the intention for the software. Formal methods help having requirements well defined from the beginning, and incremental development is a good approach for finding missing or bad requirements.

There is always code that needs to be written manually, and automatic code generation from models is something widely used these days. Its huge influence in safety needs to be

addressed, and there is a need to use qualified tools for code generation for ensuring safety.

Raising the abstraction level of languages up to formal methods was identified as an interest, and if we want to make it mainstream we need to put a clear business case to show the need and advantages of using contracts. It can reduce the amount of testing and increase the confidence level, reducing then the cost.

Closing

To conclude the discussion, the panelists were asked to provide their view on the most pressing industrial needs these days.

Ana Rodriguez insisted on the need to achieve reliability at a lower cost. Agile lifecycle models, generic reference architectures, formal methods, and dependability analysis are good ways to explore.

Ricky Sward believes that we need to give the right tools to people. More formal methods, more automated testing, and more formal processes are being introduced to improve safety.

Steen Palm said that the problem to address is how to build complex systems quickly, and it can only be done streamlining the software development process and doing verification and validation earlier. Formal methods can help a lot, although it is not widely used in the space.

The Ada Way: development of a soccer simulator

Ricardo Aguirre Reyes, Andrea Graziano, Marco Teoli, Alberto Zuccato
University of Padua, Department of Mathematics, via Trieste 63, 35121 Padua, Italy.

1 Introduction

The Ada Way is the annual student programming contest first launched in 2010 and organized by Ada-Europe [1]. This initiative is an annual programming competition among student teams where the participants have to produce a software following a project specification. The contest aims to attract students and educators to Ada in a form that is both fun and educative.

The theme for the 2010-2011 edition of *The Ada Way* was a software simulator of a soccer match [2]. The simulator had to allow playing at least a single match between two user-configurable teams. Each team was formed by players with individually configurable characteristics for technical and tactical skills (e.g. speed, fatigue, tackle skill, etc) who would play according to the team tactic. Each team had a software manager to configure the initial players line up, the initial tactic and to issue commands for tactic changes and substitutions. The simulator had to play the match according to the regular soccer rules; to do this each match must have a software referee (and eventually two assistant referees) to control the game and ensure that the applicable rules are followed.

From the architectural point of view the software system had to include a software core implementing all the logic of the simulation, one graphical window for displaying the match (and other relevant data like statistics) and two graphical panels to influence the actions of the team managers during the game.

In Fall 2010 we attended the Concurrent and Distributed Systems course of our Master Degree in Computer Science; the competition was proposed by our professor as the educational project of the course. We are a team composed by four master students at the University of Padua; we decided to participate because we considered *The Ada Way* an interesting opportunity to put ourselves at stake in an international competition that would have allowed us to prove the skills we acquired during the university course.

2 The story of a long-lasting project

Our adventure started after December 2010 when we began working on the project, shortly after the end of the Concurrent and Distributed Systems lectures. We were already aware that solving concurrent problems is a complex issue and we had the confirmation once we started addressing the project specification: we soon realized that a neat design of the simulator was a difficult challenge to achieve.

At the beginning the team was composed by only two of us, Andrea and Marco, and the original idea was not to enter the competition. As a matter of fact we began to work steadily on the project when we were close to the competition deadline. Soon after the organizers extended the deadline; at the same time Ricardo and Alberto joined the team and proposed to participate in the competition.

Figure 1: Team members with their awards. From left to right: Andrea, Ricardo, Marco and Alberto.

Our original plan was to start with the design of the core functionalities (solving the concurrent problems firstly) and then to develop the distributed features and the Graphical User Interfaces. Along with the decision to participate in the competition and with the extension of the development team we changed our original plan; in particular we decided to develop the distribution features and the GUI applications in parallel with the development of the core features. During the development of the project we made frequent partial revisions with our professor to check if our design was correct and coherent with the competition requirements. This approach allowed us to avoid incorrect solutions and to receive feedback on our main design choices. We also used prototyping to incrementally arrive at the final product. In particular we developed a non distributed version of the core system able to show the simulation on a text-based output; this prototype allowed us to test the core functionalities without waiting for the development of distributed features and GUI applications.

At the end of the development we wrote the technical report and the user manual [3] to provide a detailed description of our work, then we made the final revision with our professor and passed the exam with full marks. Finally we made the last corrections and submitted the completed work to the evaluation committee.

3 The development

Our project consisted on the design and implementation of many different components. The *Core* component is the part that actually performs the simulation of the game; it contains the implementation of the computer-driven actors (players, referees), their business logic, passive items (field, ball, ...) and any other data structure needed in order to run the simulation.

We developed three types of graphical user interfaces that can be used by humans to control the game. An indefinite number of users can act as spectators and watch the game through the *Spectator GUI* (shown in Fig. 2). Up to two users can connect as coaches, selecting and managing their team through a *Manager GUI* and at the same time watching the game using the *Spectator GUI* already mentioned. A special *Simulation user* can manage the simulation, deciding when it has to be started or stopped (thus aborting the simulation). For this purpose a *Simulator GUI* is available.

Figure 2: Spectator GUI allows users to watch the game.

All the user interfaces are distributed. None of them are required to run the simulation, because we wanted the *Core* execution to be independent from any distributed component. A time-out notifies the referee that he has to start the game in case there is no *Simulator GUI* connected, or in case nobody presses the *Start* button in time. The game proceeds without any substitution for teams that have no coaches connected. Coaches can connect during the game, anyway. Spectators are not required and their absence does not cause side effects.

In order to manage connections from an indefinite number of GUIs, we developed a component called *Billboard*, which is also distributed. For technical reasons, *Billboard* is constituted by two parts, aimed at handling communication in both directions; from *Core* to *Billboard* and vice-versa. The former module, that handles the *Core* output, is needed to run the simulation. Components can be executed on the same node (obviously) or on different nodes.

To develop this system we found that it was important to follow some design principles. The quality of design was an important evaluation criterion both for the competition and for the exam. Trying to achieve it was challenging but also necessary in order to get things work. We tried to build a modular and flexible system. Designing with separation of concerns and isolation within different components was mandatory to have a maintainable system, and to avoid unnecessary coupling that would have made the implementation harder to fix.

In the following subsections we discuss some of the most important aspects of our solution. Readers will find that there are problems which could have been addressed differently, we will try to show why we think that these ideas are good and we will mention a few other possibilities. Workable further improvements will be described afterwards.

3.1 Concurrency

Concurrency issues were the first aspects we focused on during design. Because this is an experience report of a project that had educative purposes, instead of just describing the strengths and weaknesses of the final solution, we would like to briefly discuss how we get there. As you will see, we had a long way to go.

We started by identifying entities and trying to classify them. No doubt that in soccer there are players, referees, a field and many other items, but modeling them was not always obvious. Players clearly had to be active entities, and had to synchronize themselves coherently both in time and space. The space for synchronization was the field. We have been given the good advice that things are more affordable when they are made discrete. We initially thought that the field had to be a matrix of protected objects and thus we entered the nightmare of enabling players' movements without letting them accumulate resources (i.e. field "cells"), or otherwise avoiding deadlocks.

Then we understood that it was far easier to develop an access protocol. The field was still a matrix of cells, but there was only a protected object used by players who wanted to move. The idea is that players had to declare themselves to that object (called *Barrier*) at the beginning of every single action (movement, kick, tackle, ...), and then at the end when they are done.

If nobody else is playing too close, the *Barrier* grants the right to play a move, otherwise the player task is requeued on another entry and then released as soon as the player who is too close finishes his move. "Too close" means that their concurrent actions could eventually conflict. This might happen for example when they are separated by only one free cell: if they decide to move on that cell, after having seen it free, they would make an illegal action because in our model we do not allow players to share the same position.

Time management is probably one of the most critical aspects. In our solution, we decided to split time into small time periods. The important property of time periods is that nothing is allowed to change twice in the same one. During a time period, each player can only perform a single action. All player tasks becomes ready at the beginning of the time periods in which they have to execute. The ball is a passive entity and its motion is coded in a way that it changes position exactly when a new time period begins (and never inside time periods).

The ball is the fastest object on the field. If we set its maximum speed at 30 m/s, the length of a time period is consequently fixed to 1/30 seconds. Players run far slower, so they generally execute once and then sleep for many time periods. The most important property that must be granted for the correct functioning of the simulator is that all players must have enough time to execute once in the same time period, as well as the tasks that execute in consequence of their actions. For example, when the ball falls out of bounds, also the assistant referee task wakes up to notify the referee task, which then wakes up as well to interrupt the game.

The constraint of maximum one execution per time period imposes a limit in the computations done during a short time interval. If the computations scheduled at a specific time period exceed in length, there is no guarantee that everything works properly. For example, if a player task tries to stop a ball running very fast and has only a time period to stop it[1], it would fail if the other player tasks scheduled at the same time period execute before and exceed the time slot. The player task would fail because - having no CPU time when needed - it would read the ball position too late, when the ball has already moved away. This was the worst anomaly we were able to anticipate, but we never saw it happen in reality.

The passive nature of the ball frees its movements from any delay that could have happened if the ball was an active entity. More precisely, this prevents players from delaying the ball execution, thus impeding who is eventually waiting to stop it from doing so (because the ball is still in its previous position). Its passive nature also imposes a timing constraint in execution, as described above.

We opted for this solution to achieve predictability of execution, except for some desirable non-determinism (due to the physical characteristics of players) that we developed explicitly through randomization. The simulator generates and compares pseudo-random numbers with a threshold that corresponds to the probability that a player has to do an action successfully. That probability depends on his own skills, obviously. If the generated number exceeds the threshold, the player fails (or succeeds).

3.2 Distribution

Since the first analysis and design phases, we decided to build a distributed system in order to overcome the limitations of a centralized system; in this way the whole software can run over a network and be very flexible and scalable. We proceeded partitioning each component: the *Core* that manages the simulation, the *Billboard* mediator that handles the communication between the partitions and enforces the proper constraints on the graphical interfaces, and the *Front-end* clients, which are the graphical applications by which the user can control the simulation. Each of these partitions are designed to run on separate nodes but nothing prevents from running them all on a single node.

We chose to use Java to build the *Front-ends* because we already had some experience with it, this fact forced us to use CORBA as the communication technology between the partitions, since the system was no longer homogeneous. Then we chose to adopt the Push model for both communication directions between *Core* and *Billboard*, but soon we understood that it was simpler (in terms of exchanged data types) to use the DSA and consequently to develop a homogeneous *Core-Billboard* subsystem. A different choice was made in terms of the exchanged data between *Billboard* and the Java *Front-ends*: we decided to use the Publish-Subscribe paradigm because we had some different types of GUI, hence different classes of data consumers. Subsequently, during the implementation, we came across some hurdles that forced us to revise the design and to adopt different strategies and solutions. In the following paragraphs we provide a brief description of the main obstacles that we encountered during implementation; things seems always well-defined in theory but when one goes deeply into practice realizes that the path is not so straightforward.

One of the first issues regarded the interface between *Billboard* and the *Front-ends*: we began to write the IDL data structures that had to be used as objects exchanged between the two partitions. Once compiled these IDL files in Java and Ada, we saw that the code generated by IDL was really similar to objects that Ada uses in its remote procedures calls, so we tried to use the same Ada data structure generated by the IAC compiler, using them as objects parameters. We realized that IAC generates data structures based on CORBA types and those were not the same types used by the DSA (needed for *Billboard-Core* communication). We tried to define some Ada remote types using CORBA types but we had no success.

We also had some problems configuring PolyORB: once downloaded and installed the software using "make install" like any other program, the installation completed without errors, but when we tried to run PolyORB examples we saw that something was wrong. We studied the Gnatdist macro-compiler documentation, trying to understand why it did not work. Eventually we wrote on Ada IRC channel for help and they gave us the correct installation parameters revealing that the configuration step was not so trivial.

During the process of building and interfacing the GUI, we read a lot of Java documentation and examples, noticing that Java only implements a small part of CORBA specification. "Event Service" and "Notification Service" in particular are not implemented; all examples were outdated, prompting us to look for other CORBA implementation providers. We found that nobody has given continuity to the old open-source projects like OpenORB; in the industry Oracle practically bought all the other vendors like BEA Tuxedo and Glassfish-CORBA. We understood that we were dealing with a market monopoly issue, so we wrote on the PolyORB mailing list asking if anyone had been able to use the "Event Service" between Java and PolyORB before. They answered:

> "The biggest problem is that even though all vendors have good intentions and believe in the seamless interoperability between different products, whenever something bad happens on the line between two unrelated implementations, you will

[1] In order to stop the ball, the player must be in its same cell

> find it difficult to get all those vendors together to help you."

Maciej Sobczak recommended us to use YAMI4; we read that it was presented to Ada-Europe last year, so we decided to give it a try.

We installed YAMI4 and tested it on Java, this time without troubles, it worked fine like any other API. Regards Ada it was a little bit different because at first we had to solve a library linking problem. Having written to the YAMI4 mailing list, they recommended us to include all YAMI4 sources within our project, because otherwise we would have got always the old linking problems. We did not like that solution but we did not know how to solve the problem otherwise. At the end we had other little issues like synchronization that were solved quickly.

Once the development of the modules was at a good stage and we were ready to integrate the parts, we stumbled across some integration problems. Firstly we had some equivalent types defined in both Ada partitions, so we resolved combining and placing them inside *Billboard* partition. Then we had to face a circular dependency problem: when the *Core* started up, it needed to have *Billboard* already started and when *Billboard* started up needs to have the *Core* already started. So we decided to split *Billboard* in two partitions: *BillboardInput* and *BillboardOutput*.

Finally, when we integrated the whole *Core* within the distributed project we had many dependency problems; to solve them we needed to define a common directory included by *BillboardOutput*, *BillboardInput* and *Core* Ada projects. The fastest solution (clearly not the best) was to include all directories in all projects.

Apart from technological issues, we learned that distribution brings many additional difficulties compared to the classical way of coding a centralized system. Working on separated nodes exposes to delays because there is the network in between; moreover we have to deal with possible network issues and a whole new set of problems such as equivalence of data types, caching policies, load balancing and the cycle of operation of the system. Those are things that have to be tested during system integration.

Sometimes distribution directly affects the design of the internal parallelism of the partitions, so we need to build robust protocols and well-defined interfaces in order to have a coherent and maintainable system and avoid or mitigate exceptional dynamic situations that could arise.

3.3 The technical specification and COMET

The technical specification of our project is also the report we submitted for the exam at the university. That specification contains an extensive description and discussion of almost all the choices we made during the development phases. We started writing the report when we were fixing and connecting together the distributed components of the simulator.

During the development, we found a book that presents a methodology for the development of concurrent and distribution systems called COMET [4]. That book presents an extensive set of case studies.

At that time it was too late to follow the methodology because we had already made many design choices, but we decided to use the book to shape our report according to its case studies. We thought that this extra effort would have helped us to structure the report (by giving us a skeleton) and that it would have been instructive to analyse the quality of our work from a different perspective.

Re-thinking the project following COMET would have required us an important effort and would be time consuming, because we did not start with it from the beginning. We spent more time than what we might have saved using its use cases. Thus it has not been an advantage for our report, but it has been instructive because we had to analyse examples of good design and to compare them with our project.

In its first half, our specification is really technical and adherent with COMET. In its last half, it becomes more descriptive because we wanted to provide an in-depth discussion for all the design choices that were more related to the topics covered during the course.

4 Things we left behind

During the development phase we had to look for some trade-offs. This was partially due to the huge size of the project that required us to cut off some aspects. On the other side, some aspects could have been addressed differently; in some case we selected a solution according to some criteria (e.g. determinism/non-determinism) and in some other we found many solutions and then we tried to choose the best one. This was not trivial because in many cases there are many choices that are equally good, while in others we had not clear how to choose, due to lack of experience. Sometimes experience enables the intuitions needed to make the right choices, and gives some knowledge that cannot be obtained from books.

It also happened that we thought "Maybe we could have done/tried doing this way...", but still without the certainty that it would have been better and clearly without the time to try many different alternatives. In this section we present some good argument to argue on our choices and to show that there is room for improvements, starting from the "Barrier" mechanism which is probably more interesting than others for whom (like us) are more concerned with concurrency and distribution issues than with other aspects.

Since all the awaiting players are held on the same queue, they all have to be re-evaluated when someone else ends, and then requeued if the ending player was not the one who prevented the player under evaluation from playing (the entry closes automatically when they have all been re-evaluated). This single queue for awaiting players is clearly a bottleneck, because in order to release a player we re-evaluate all those that are waiting. A possibility would be the use of a family of entries. We did not immediately realise this possibility and then we found not trivial to find how to adapt our solution accordingly. The potential advantage of doing so is anyway reduced by the fact that players are not generally close to each other, nor they always get ready at the same time.

Due to their business logic, players are not generally close to more than one opponent at a time (and they are typically not close to teammates). This means that in our expectations, many player tasks are generally allowed to execute concurrently. Because of time management, as we will see, a small number of them becomes ready at the same time, so a few of them are actually requeued.

Other improvements can be done on players' artificial intelligence. Players can be seen as reflex agents: they see the external environment, they choose what to do following the condition/action rules given by their business logic and then they act accordingly. Their business logic is quite simple and has a huge room for improvements. Just to give an example, when they pass the ball they do not check the positions of their opponents, they just look for the closest teammate whose position is more closed to the opponents' goal. If there is an opponent in between, they do not change decision.

Improvements can be done also in how their physical characteristics are used. They have a level of health that decreases when they are tackled and decreases more if they receive fouls. If their health - after a foul - gets too low, they get injured and leave the field. The health mechanism could be far more sophisticated in order to approximate fatigue in a better way. For example, the physical characteristics of players (speed, power, accuracy, ...) could be decreased according to their health. This would be quite easy to implement (because the simulator already supports this feature), but it is probably time consuming to tune the system accordingly, because it requires to test the simulator with many different values until the right ones are found.

Other possible improvements are the support for a series of matches, a better support for the physics of the ball and a more realistic simulation. Players currently pass the ball impressing it a fixed angle, in a way that the ball hits the ground for the first time in a position closed (depending on their accuracy) to that of their target teammates. There is no drift, but at each bounce the ball loses some energy, until it stops. When trying to volley the ball, players never hit it badly; this is unrealistic, because it should be common for them to cause deviations or to interfere with the ball without being able to give it a proper direction. Similarly, when a goalkeeper is not able to catch the ball but he is able to reach it with his hands, he can hit the ball to throw it away. We are not currently simulating these behaviours, but they would improve the quality of the simulation. Corner kicks for example are never assigned because of this.

Regarding distribution, we implemented all the requirements and the room for improvement is limited. In fact, having decided at the beginning the key design factors, the rest of the work proceeded smoothly, apart from technical configuration troubles with the tools that we used. Certain things however could be improved, like the definition of the messages; having a standard structure among various classes of messages in fact helps to minimize the misinterpretations of their data fields. Other improvements could be made on certain functionalities on user's side, to increase the usability of the GUI. As a rule of thumb we saw that giving less constraints to the user results in more complicated code, that's why we chose the simpler but still effective solution.

5 Conclusion

Ada is not just a programming language, it is a world and has its own way to do the whole. Ada perfectly suits the *iceberg analogy* because it is not just about learning the language: beneath the surface you see there is much more that you need to know. Ada has its own, clever and unique paradigm. Its learning curve seems pretty steep and it is common to have hard times before mastering Ada, but at the end it's worth to learn its good style.

Sometimes during the project we experienced the difficulty of finding out someone who found and possibly solved our own problems, which is so much easier with more commonly used languages. From this perspective Ada looks like a language for the chosen few, not supposed - and probably not willing - to conquer the masses.

Working in a group always requires a relevant time overhead for organization and communication. This is the price to pay, which is even higher if you do not have a long teamwork experience. Anyway we think that the decision of creating a team was essential for our success; at the end we think that we achieved our goal because we were really motivated in finishing the project. At the beginning everybody wanted to finish as soon as possible, and in the last days we strived to complete the project as good as possible.

The project was very challenging, but it also took us much time. In our opinion the requirements were oversized. It was time expensive to develop a full soccer simulator and it has not been easy to fulfill all its requirements. Now the project has already entered the "try and beat me" mode and we hope that students will find it easier to keep their time and effort under control with this modality. By the way, they will start from our solution, making it better and better. We are honoured that our solution has been chosen as the reference implementation and we will be pleased, hopefully, to see the improvements that could come from the forthcoming competitions.

Even though it has been time expensive, this project was a worthwhile part of our education that allowed us to learn many advanced aspects which are essential to enrich our background. Finally, we believe that the strong effort required was rewarded by the achievement of an outstanding result.

References

[1] Ada-Europe organization. http://www.ada-europe.org

[2] Student programming Contest "The Ada Way". http://www.ada-europe.org/AdaWay/

[3] Project documentation and source code. http://www.ada-europe.org/AdaWay/Reference_Implementation.zip

[4] H. Gomaa (2000), *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison Wesley.

Combining Code Generation and Ada Generics to implement a Software Product Line

Frank Dordowsky

ESG Elektroniksystem- und Logistik-GmbH, Livry-Gargan-Str. 6, D-82256 Fürstenfeldbruck, Germany; Tel+49 8 9216 2871; email: Frank.Dordowsky@esg.de

Richard Bridges, Dr. Holger Tschöpe

Eurocopter Deutschland GmbH, D-81663 München, Germany; Tel:+49 89 607 28891/+49 89 607 26713; email:{Richard.Bridges, Holger.Tschoepe}@eurocopter.com

Abstract

Distributed embedded real-time systems such as the NH90 avionics system exchange a large amount of data over real-time data-buses. Due to space and time restrictions, the data in the messages are highly compressed and represented at the bit level. Moreover, system variants usually constitute varying sets of equipment connected to the data buses so that the data transmitted via the data buses are highly susceptible to system variants.

The real-time application software must decode and encode the data contained in the messages for processing. This software is complex and difficult to develop, especially with the additional requirement that it must be highly adaptive to support varying system configurations.

As a solution to this problem, the software architecture of the main computers of the NH90 avionics system provide a generated high level interface of the message data to the application software, where the data is represented in domain specific Ada data types rather than in raw bit format.

This paper shows how this IO data interface is designed, and how it is generated using Ada generic units as basic building blocks. It also describes the tools used for generation and the approach taken to qualify the generator tool suite.

Keywords: Avionics Systems, Software Product Lines, Ada

1 Introduction

1.1 Project background

The NH90 is a medium weight multi-role military helicopter that comes in two basic versions: the Tactical Transport Helicopter (TTH) and the NATO Frigate Helicopter (NFH). It is being produced in more than 20 variants for 14 nations and their armed forces.

The software division at Eurocopter Germany develops the on-board software for three computers of the NH90 avionics system. The growing number of customers and their specific set of mission requirements for the NH90 have led to an increasing number of functionally different helicopter variants. In order to cope with the high number of software variants, the NH90 software team has developed concepts and strategies for SW architecture and tool modifications based on Software Product Line (SPL) principles [1]. The implementation of the NH90 SPL relies on three pillars [2] software architecture, software design pattern and, to a very large extent, code generation.

1.2 Avionics system architecture

The NH90 avionics system architecture is a typical representative of the federated avionics architecture. In contrast to Integrated Modular Avionics (IMA), *federated* architectures are characterised by a multitude of specialised equipment that implement the avionics functions. This equipment (also called *devices* in NH90 terminology) is connected via avionics data buses to a central main computer that also controls the deterministic message traffic on the data buses. Typical examples for avionics data buses are MIL-Std 1553B (Milbus), ARINC 429, AFDX, but also serial lines (RS-232, RS-485). For reliability and safety reasons, the main computers as well as the avionics buses and even critical equipment is often dual redundant.

The NH90 avionics architecture is structured into two large subsystems, each controlled by a dedicated main computer: the *CORE Management Computer* CMC controls the CORE subsystem, and the *MISSION Tactical Computer MTC* is responsible for the MISSION subsystem.

The crew interacts with the system with *Multi-Function Displays (MFD)* and *Display and Keyboard Units (DKU)*. MFDs provide system, flight and mission data in a graphical format to the crew, whereas DKUs are used for displaying and keying in alphanumeric data [3].

Main computers, MFDs and DKUs as well as most equipment contains micro processors, so that the overall system forms a distributed, embedded real-time system.

Avionics systems and military mission systems are safety critical – for the NH90, a project specific safety level was

defined that ranges between design assurance level B and C as defined by DO-178B [4,5].

NH90 product variants are established by different system configurations, i.e. different set of equipment of equipment capabilities in case an equipment is itself configurable. There are also a varying number of DKUs and MFDs, and different sets of equipment lead to variations in the data to be displayed on DKU and MFD. Especially the data transmitted via the avionics data buses differ largely between variants.

1.3 Messages and Signals

A message is a set of data words that are together transmitted over an avionics bus at one time. Size and number of data words are different for the various data bus technologies: A Milbus message can consist of up to 32 data words of 20 bits each, with a payload of 16 bit. ARINC-429 buses transmit single labels that consist of a 32 bit data word with a 19 bit payload.

A signal is the smallest unit of data to be transmitted over the data bus. It may encompass of a single bit only but can also extend over several data words (e.g. in case of strings). For every signal, the Most Significant Bit (MSB) and the Least Significant Bit (LSB) must be specified to indicate the order in which the signal representation shall be interpreted.

In the NH90 project, many signals have a *validity bit* attached to it, to indicate to the processing unit that the transmitted data is valid or not.

The NH90 avionics main computers must handle a very large amount of signals as indicated in Table 1:

Table 1- Number of Avionics Bus Signals

Avionic Bus Signals	CMC		MTC	
	IN	***OUT***	***IN***	***OUT***
MIL-1553	13500	26600	5500	7000
ARINC 429	500	1200	300	2700

The CMC must handle a total of more than 40,000 MIL-Std 1553B signals and the MTC about 3,000 ARINC 429 signals.

The signals are typed: *signal data types* that are used to define the semantics of the signals at system level. System engineers use a data base tool to define and maintain messages, signals and their types (see section 3.1). Table 2 shows the signal data types that are used within the NH90 project.

The definition of a numeric interface type (i.e. one of BIN, BN2 and BCD) for example includes the following information:

- the number of bits used for the presentation of the signals of that type,
- LSB and MSB,
- the value represented by the LSB, i.e. the precision of the signal values,
- the minimum and maximum value for that type,
- and a default value.

This type of information is captured into a data base tool by systems engineers during systems design and later used by the code generators.

Table 2 - Signal Data Types

ID	Signal Data Type
AS7	ASCII 7 String
AS8	ASCII 8 String
BIN	Unsigned Binary
BN2	Signed Binary (2's complement)
BCD	Binary Coded Decimal
TOR	Boolean
DIS	Endumeration

1.4 Message Data Processing

The application software on the main computers must extract the data from the messages in up to 50 Hz cycles, process the data and encode them into messages to the equipment. Due to different structure, formats and sizes of the signals, the software for decoding and encoding the data is complex and therefore difficult and error prone to develop. Moreover, since the system variants have a large impact on the data of the avionics buses, the software for decoding and encoding must be very adaptive to support this variability.

Our solution to this problem is to provide a high-level Ada interface to the application software with Ada types representing domain level concepts rather than bit encoding. Decoding and encoding of the message data is handled by the implementation of this interface. In order to manage the variability for the NH90 system variants, the interface and its implementation is completely generated.

The remainder of the paper will describe the architecture of the I/O subsystem and how it fits into the overall software architecture, the tools used to generate the software, and finally the approach taken to qualify the code generators.

The problem addressed by this paper is common to distributed embedded real-time systems and so that the solution may be of interest to other developers of similar systems. The solution can also be transferred to other languages than Ada as long as they provide a mechanism similar to Ada generics.

2 Software Architecture

Code generation is strongly related to the overall structure of the software. This section provides an overview over the software architecture of the NH90 main computers that is necessary to understand how low-level message data are transformed into high-level data and vice versa.

Both CMC and MTC share the same hardware and the same software architecture. The architecture of both

computers is shaped by the *embedded real-time framework NSS*.

2.1 Embedded Real-time Framework NSS

The software framework NSS (NH90 System Software) relieves the application programmer of intricate real-time programming tasks such as real-time scheduling, error and exception handling, redundancy management, and, as the subject of this paper, device and I/O handling as well as the data conversion between Ada data structures and raw I/O data.

The main components of the architecture are the following (Fig. 1):

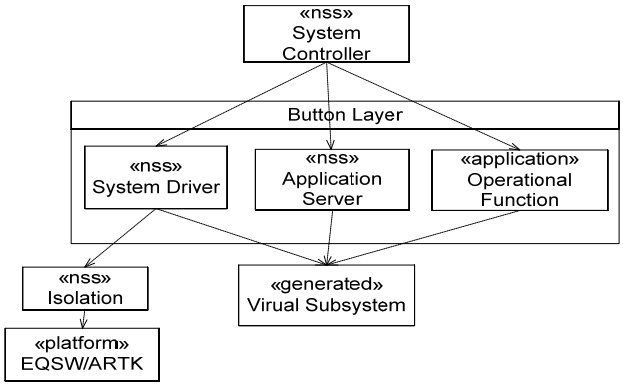

Figure 2: Software Architecture Overview

The **System Controller** forms the uppermost layer and controls the complete operational software. It is responsible for the control, schedule and dispatch of Ada tasks as well as general management functions such as start-up, shutdown and redundancy management for example. The system controller also determines the helicopter variant by reading a designated non-volatile memory location that contains the variant id string.

The **System Drivers** drive the physical exchange of raw I/O data. There are drivers for Milbus, ARINC, Ethernet, serial and discrete lines. They store data received from external equipment in the *Virtual Subsystems* (see below), or retrieve data from the Virtual Subsystems and send it to the external equipment.

The **Application Servers** make use of the system drivers and provide a higher level of abstraction on data exchange. Important examples of Application Servers are the *DKU Server* which handles the communication to the DKUs, or the NVM Server which manages access to the non-volatile memory (NVM) of the main computers. Another application server that is important for the handling of I/O data is the *SMD server* which is explained in more detail section 2.2 below.

The **On-board Processing Functions (OPFs)** implement the proper application specific processing requirements. An OPF is a set of operations which accept data or control input, perform computational or control functions and produce data or control output. The OPFs form the application components and are referred to as *operational software*.

The **Virtual Subsystems** are a common real-time data store which provides a uniform data abstraction of the avionic subsystems to the operational processing functions, and isolates these from I/O interface specific data representations. They also organise the data exchange between application components and between application components and the NSS. Virtual subsystems represent typical avionics subsystems such as Navigation, Communication, Mission Management, etc.

The **Isolation layer** provides an abstract interface to the equipment software (EQSW) and the Ada Run-time Kernel (ARTK). It isolates the operational software from system dependent features and enables the operational software to run on target and host platforms as well as the different hardware architectures with only minor modifications.

The **EQSW/ARTK layer** contains the software provided by equipment manufacturers of the computer hardware and the compiler vendors.

The major topic of this paper is the interface between the on-board processing functions (the operational software) and the virtual subsystem as it is detailed in section 2.3 below.

2.2 I/O Processing

The original hardware of the NH90 main computers consists of two separate processing boards and shared memory between them, as shown in Fig. 2.

The IO processing board handles all input and output of computer, whereas the data processing board executes the application software and the control logic. There are two independent executables for each processing board, so each main computer forms a distributed real-time system in itself.

The system drivers, running on the IO board, obtain messages from the hardware interfaces via equipment software and place the messages into a *global buffer* in shared memory that is used to exchange data between the two processor boards.

The *Shared Memory Data (SMD) Server* takes the messages from the global buffers and places them into the *raw buffers*. Up to this point, messages are handled as array of data words without considering their internal structure.

The raw buffers are aligned with the *formatted buffers* using a well-known overlay technique. The virtual

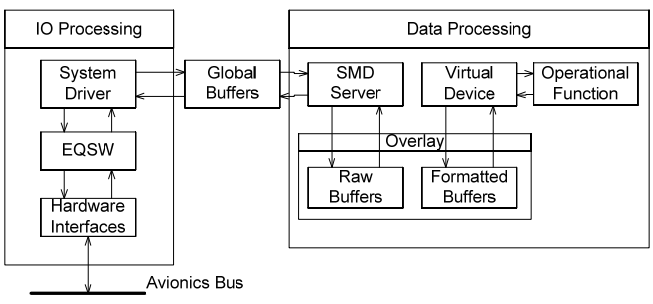

Figure 3: - I/O Processing

subsystems access the formatted buffers to provide the operational software with a high-level view on the data.

The way back is similar: the operational software writes outgoing data to the formatted buffers using the high-level interface provided by the virtual subsystems. The SMD server reads the data for the outgoing messages from the raw buffers and places them into the global buffer. The system drivers then pick up the messages and transmit them via equipment software and interface hardware to the devices that are connected to the avionics data bus.

2.3 Interface between Virtual Subsystems and Operational SW

The interface between the virtual subsystems and the operational software provides the high-level abstraction of the message data as indicated in section 1.4 above. Virtual subsystems are divided into virtual devices – a virtual device is an Ada package representation of a real equipment connected to a data bus, or an equipment capability if the equipment is configurable. The virtual devices form one of the basic building blocks of the NH90 SPL.

Since a virtual device is implemented as an Ada package, the Ada package specification of the virtual device represents the high level interface to the message data to and from that device. The following code section shows a part of the Ada package specification for the FLIR (Forward Looking Infrared) virtual device:

```
with ...
package Flir_Vd is
   …

   procedure Get_Flir_Los
         (Value : out Flir_Los;
          Valid : out Data_Validity);

   procedure Put_Flir_Mode_Cmd
         (Value : in Flir_Mode_Cmds);
   …

end Flir_Vd;
```

The name of the equipment is encoded in the Ada package name.

Domain specific high-level data structures are represented by *Software Objects*. A scalar software object consists of a single data item that represents a single signal. In addition, the programmer can define more complex structures to represent more sophisticated domain objects, although not the full range of Ada types are admitted for complexity reasons – only the following structured software objects are possible: record types, array types, matrix types (two dimensional arrays), and arrays of records. Nesting, i.e. records of records, is not possible.

For every software object, there may be a *put procedure*, or a *get procedure*, or both, depending on the direction of the messages that eventually contain the data of the software object. A put procedure provides data to the drivers to be transmitted to the equipment; a get procedure allows retrieval of the data that has been received from the equipment.

In the code section shown above there is a get procedure for the Line-of-Sight (LoS) Data that are actually a record containing azimuth and elevation of the line of sight of the FLIR sensor. The validity of the LoS data is indicated with a separate Valid parameter in the get procedure.

The FLIR equipment accepts moding commands that are also modelled as software objects so that they will be transmitted to the FLIR when placed into the virtual device with the corresponding put procedure. The type for this software object is an enumeration type. The NH90 coding rules force all enumeration types to include the enumeration literal `Undefined` that indicates non-validity so that the put and get procedures for software objects with enumeration type do not require an additional parameter for the validity.

To summarise: the high level Ada interface of message data is contained in a number of virtual devices that are part of a virtual subsystem. The virtual devices represent a real equipment connected to one of the avionics data buses. As such, the virtual devices are an instance of the proxy pattern [5,6]. The high-level data of a virtual device is represented as software objects, and for every software object there may be a put and a get procedure. On-board processing functions use this interface as a high level interface to low-level I/O data without the need to know about their bit-level representation.

The following subsections describe the implementation of the virtual subsystems in more detail.

2.4 Virtual Subsystem Architecture

The virtual subsystem component of the architecture is in itself layered in order to reduce the complexity of the generated code and to facilitate debugging and qualification of the generated code. Fig. 3 shows the three sub-layers:

1. The bottom layer is the *Generic Sub-layer* that provides a fixed set of fundamental translation generics for numeric data, enumerations and strings.
2. The *Conversion Sub-layer* is a completely generated set of the instantiations of the translation generics for all required mappings between low-level message data and their high-level Ada representations.
3. The *Structure Access Sub-layer* provides the actual access to the high-level virtual subsystem data, as well as more complex data structures such as arrays and records. This sub-layer is also completely generated.

Aside to these sub-layers, there is a generated data component that contains the message buffers. The raw message buffers as they are used by the system drivers are simply arrays of data words. They are overlaid with the

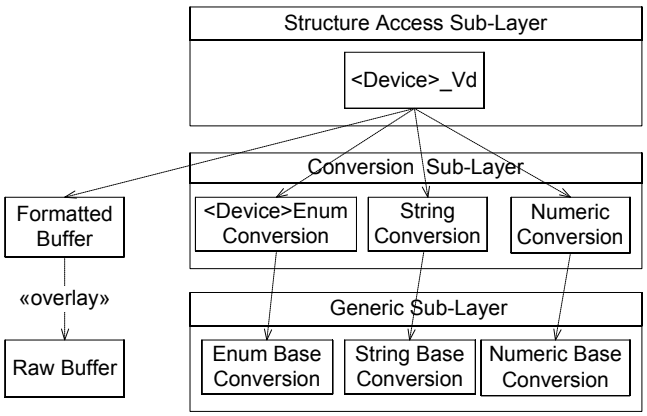

Figure 4: VS Layers and Components

formatted buffers that represent the bit level layout of the messages.

2.4.1 Buffer Overlay

There is a separate buffer for every message on the data buses. These *raw buffers* are aliased with record structures as shown in the code section below: the formatted buffer `Flir_Los_Fmt` is aliased with the corresponding raw buffer `Flir_Los_Raw`. The overlay is enforced with an address specification.

```
Flir_LoS_Fmt : Flir_LoS;
   for Flir_LoS_Fmt  use at  Flir_LoS_Raw'Address;

   type Flir_LoS  is
      record
         ...
         Flir_LoS_Valid : T_B_Validity;
         ...
         Flir_Los_Az : Raw_Int_16;
         Flir_Los_El : Raw_Int_16;
      end record;

   for Flir_LoS use
      record
        ...
        Flir_LoS_Valid at 0 range
          W16_Bit_2_2'First .. W16_Bit_2_2'Last;
        ...
        Flir_Los_Az at 2 range
          W16_Bit_0_15'First .. W16_Bit_0_15'Last;
        Flir_Los_El at 4  range
          W16_Bit_0_15'First .. W16_Bit_0_15'Last;
      end record;

   for Flir_LoS'Size use 48;
```

The record components of the formatted buffer refer to the individual signals of the message that will be put into the corresponding raw buffer. The component types form an Ada representation of the signal data types - they are called *Driver Basic Types.*

Generated record type presentations allow a bit level access to the individual signals within a message buffer. The position and length of the individual components are calculated by the generator tools from signal type information that are associated with the signal (see section 1.3).

The ranges of the component locations are specified indirectly with the `'First` and `'Last` attributes of a type that represents the bit positions within a data word. This is necessary because the project uses several compilers of different vendors, and these compilers define a different bit and byte ordering in a data word. Since Ada 95 it is possible to enforce the bit ordering of a type with the `Bit_Order` attribute, but this was not available with Ada 83.

2.4.2 Generic Sub-layer

The generics sub-layer provides the basic building blocks for the data conversion in form of a set of generic transformation routines. There is a total of 8 generic conversion packages called *Base Conversions*. The base conversion packages are required for the combination of interface types and high-level software types, as indicated in table 3. There are three classes: *numeric conversions, discrete conversions,* and *string conversions*. The interface types are those defined in section 1.3 above. For the numeric conversions, the target software type is either an integer type or a floating point type.

Table 3 - Generic Conversion Packages

Class	Generic Conversion Package (Base Conversions)	Interface Type	Software Type
Numeric	Bin_Int_Generic	BIN/BN2	Integer Type
	Bin_Real_Generic	BIN/BN2	Real (Float) Type
	Bcd_Int_Generic	BCD	Integer Type
	Bcd_Real_Generic	BCD	Real (Float) Type
Discrete	Bool_Generic	TOR	Boolean
	Enum_Generic	DIS	Enumeration
String	Str_Ascii7_Generic	ASCII7	String Type
	Str_Ascii8_Generic	ASCII8	String Type

The code snippet below shows the generic package specification for the binary to real conversion as an example.

```
generic
      type Raw is range <>;
      Raw_Min : Raw;
      Raw_Max : Raw;
      Raw_Default : Raw;

      type Eng is digits <>;
      Eng_Min: Eng;
      Eng_Max: Eng;
      Eng_Default : Eng;
      Lsb : Universal_Types.Real;

package Bin_Real_Generic is

      type Image_With_Val is
         record
```

```
            Value : Eng;
            Valid : Boolean;
        end record;

    type Raw_With_Val is
        record
            Value : Raw;
            Valid : Boolean;
        end record;

    function To_Image
          (Source : in Raw;
           Validity : in Boolean)
     return Image_With_Val;

    function To_Raw
          (Source : in Eng;
           Validity : in Boolean )
     return Raw_With_Val;

end Bin_Real_Generic;
```

The package imports the drivers basic type (`Raw`) and the high-level Ada type (`Eng`), together with their admissible minimum and maximum values. The last generic formal parameter declaration `LSB` represents the precision of the representation.

The generic package exports two conversion functions, one for the conversion of the raw driver type to the high-level Ada type (function `To_Image`) and vice versa (`To_Raw`).

The conversion functions do not only convert the values but also establish the validity of the converted values. Two record types that bundle the converted value with its validity are necessary to be able to use conversion functions rather than procedures.

As shown in the code snippet below, the actual conversion is performed in the declaration part of the conversion function. The body of the function then checks if the converted value is within the admitted range and returns not valid (`Validity = False`) if this is not the case.

```
function To_Image
      (Source : in Raw;
       Validity : in Boolean ) return Image_With_Val is

   Result : Image_With_Val
     := (Value => Eng_Default, Valid => False);
   Calculated : Universal_Types.Real
     :=  Universal_Types.Real (Source) * Lsb;

begin
   if Calculated in Eng_Min .. Eng_Max then
      Result
        := (Value => Eng (Calculated), Valid => Validity);
   else
      Result := (Value => Eng_Default, Valid => False);
   end if;
   return Result;
exception
   when Numeric_Error | Constraint_Error =>
      return (Value => Eng_Default, Valid => False);
end To_Image;
```

All other conversion functions in the eight generic base conversion packages are set up in a similar way.

2.4.3 Conversion Sub-layer

The conversion sub-layer consists of a large set of generic instantiations of the base conversion packages and is completely generated. The generator exploits the type mapping between the interface type and the drivers basic type on one hand and a similar mapping between the interface types and the high-level software types on the other (see Fig. 4).

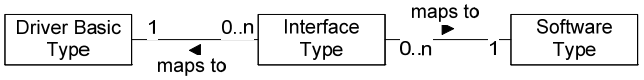

Figure 5: Type Mappings used for Generation

These mappings are managed by software engineers in a data base tool called ODIN (see section 3.1 for more information on ODIN). For every interface type there is a pair of driver basic type and software type, and for every type pair there is an instantiation of the suitable base conversion package.

The code snippet below shows an instantiation of the binary to real conversion for the conversion of an angle used in the MISSION system:

```
package T_I _Angle is
  new Bin_Real_Generic
        (Raw => Raw_Int_16,
         Raw_Min => Raw_Int_16 (-32768),
         Raw_Max => Raw_Int_16 (32767),
         Raw_Default => Raw_Int_16 (0),
         Eng => Degree,
         Eng_Default => Degree (0.0),
         Lsb => Universal_Types.Real (0.00549317));
```

The instantiation uses a general purpose floating point type `Degree` exported from package `Universe` that represents angles measured in degrees.

2.4.4 Structure Access Sub-layer

The Structure Access Layer now implements the operational interface, i.e. it provides the package bodies of the package specifications introduced in section 2.3 above. The code section below shows the implementation of the body of the get procedure for the `Flir_Los` software object, which is actually a record with the components `Flir_Los_Az` (Azimuth) and `Flir_Los_El` (Elevation). The implementation uses the conversion package for angular data that was shown in section 2.4.3 above.

```
package body Flir_Vd is

  procedure Get_Flir_LoS
        (Value : out Flir_LoS;
         Valid : out Data_Validity) is

    My_Los_Az : T_I_Angle.Image_With_Val :=
```

```
            T_I_Angle.To_Image
              (Flir_LoS_Fmt.Flir_LoS_Az,
               Flir_LoS_Validity and
              To_Bool
               (Flir_LoS_Fmt.Flir_LoS_Valid));

        My_Los_El : T_I_Angle.Image_With_Val :=
            T_I_Angle.To_Image
              (Flir_LoS_Fmt.Flir_LoS_El,
               Flir_LoS_Validity and
              To_Bool
               (Flir_LoS_Fmt.Flir_LoS_Valid));
    begin
      Value := (Los_Az => My_Los_Az.Value,
                Los_El => My_Los_El.Value);
      Valid := My_Los_Az.Valid and My_Los_El.Valid;
    end Get_Flir_LoS;

  end Flir_Vd;
```

The actual conversion takes place in the declaration part, mainly because the NH90 coding rules require all local variables to be initialised. This is possible because the conversion subroutines are declared as functions and not as procedures. The input parameters to the conversion functions are the components of the formatted buffer that represent the signals of the message that encode the azimuth and elevation, as well as the signals validity. Note that the overall validity of the component of the software object is the logical conjunction of the signal validity and the overall message validity (`Flir_Los_Validity` is the message validity and `Flir_Los_Fmt.Flir_LoS_Valid` is the signal validity).

The structure access sub-layer is completely generated and can become quite complex for software objects of more sophisticated types such as matrix of array of records.

3 The NH90 Tool Chain

3.1 NH90 Code Generation Tools

Code generation in the NH90 software project is based on data kept in relational databases rather than in graphical models or domain specific languages (DSL). The advantage of this approach is its capability to consistently handle a very large amount of data (more than 10,000 signals, see section 1.3). Fig. 5 shows the tools that are used for code generation [1]:

The definition of equipment, equipment interfaces together with messages, their signals and the signal data types are maintained by systems engineers in a database tool named *Avionics Data Base System (ADBS)*. These data are imported over a gateway into the software engineering database *ODIN (OFRS Data and Interfaces of NH90)*. With this import, the software engineers have the information described in section 1.3 available so that it is not necessary to re-enter them into their tools. The Software Requirements Specifications (SRS) are created by systems engineers in the requirements engineering tool DOORS and exported as documents. The software

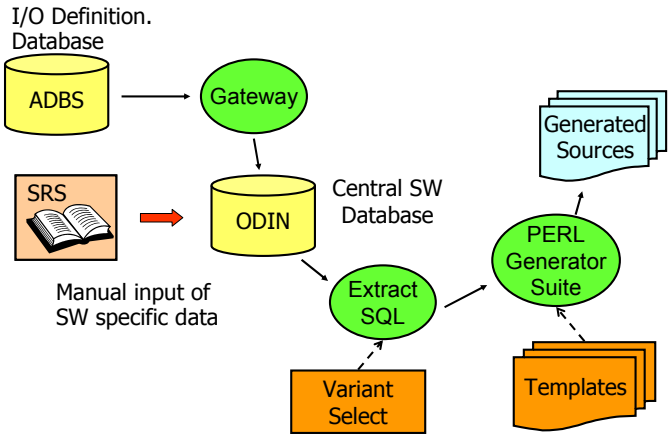

Figure 6: NH90 Code Generation Tool Chain

engineers use the SRS to establish high-level domain types. In ODIN, the software engineers map the low-level driver basic types and the high-level domain abstractions to the signal data types that are imported from ADBS, as indicated in Fig. 4 in section 2.4.3 above.

Extract scripts export selected data from ODIN to prepare for subsequent code generation. Since not all variants need all I/O data, the extract scripts take a selection of helicopter variants as input and select only those data from ODIN that is relevant to the selected variants. This is described in more detail in section 3.3 below.

The generator suite is a set of PERL programs that take the data extracted from ODIN and uses a set of templates to generate the final source code, which can then be compiled and linked to the operational software. The generator suite is not aware of helicopter variants – all variant specific processing and selection is performed in the extract scripts, solely controlled by data of the ODIN database.

3.2 Template Based Code Generation

The generation process is based on templates specified in the Perl template language [8]. The following listing shows the template for the generation of the virtual device package specification, using directive tags of the *Template Toolkit*:

```
<tmpl_include name="Common/spec_header.tmpl">
-------------------------------------------------------
--  Package <tmpl_var name="device">_Vd
--  provides the procedural interface for
--  <tmpl_var name="device"> Vd.
-------------------------------------------------------
<tmpl_loop name="withs"><NOBR>
with <tmpl_var name="name">;
</tmpl_loop><NOBR>

package <tmpl_var name="device">_Vd is

<tmpl_loop name="sw_object">
  -------------------------------------------------------
  --  This represents the procedural interface
  --  to the <tmpl_var name="suffix"> VD.
  -------------------------------------------------------
```

```
    <tmpl_if name="is_get"><NOBR>
      <tmpl_include name="SS#LLD.ads.get.tmpl">
    </tmpl_if><NOBR>
    <tmpl_if name="is_put"><NOBR>
      <tmpl_include name="SS#LLD.ads.put.tmpl">
    </tmpl_if><NOBR>
  </tmpl_loop><NOBR>

  end <tmpl_var name="device">_Vd;
```

The template variables such as `device` or `sw_object` will be substituted with data obtained from the extract files. The template in the listing contains a loop over all software objects associated with the device (`tmpl_loop name="sw_object"`). Template control logic (e.g. `tmpl_if name="is_get"`) checks if there is a get procedure defined for the software object, or a put procedure or both. The generation of the actual procedure specification is delegated to another template using the `tmpl_include` tag.

With the sophisticated control logic of the template language it is possible to control the output of the generation process using the data extracted from the database tool ODIN.

3.3 Variant Handling in the Generator Tool Chain

One of the driving forces for the NH90 SPL is that the software for a helicopter variant or set of variants shall only include the code that is necessary for that variant or set of variants, and not more [1,9]. Moreover, the software for a helicopter variant or set of variants shall be assembled from pre-fabricated components rather than copied and modified, which also applies to data used for code generation. To achieve this in an efficient way, the generator tool chain has to fulfil the following requirements:

- Selection and control of variant specific code only from data kept in the ODIN database (data centric rather than code centric).
- A single data repository for all variants in order to avoid duplicated effort for maintaining several copies of the same data. This comprehensive repository is called a *superset*.

The solution to these requirements is the identification of building blocks that can be assembled to form a helicopter variant. Virtual devices are the ideal candidates to become these building blocks because software objects are too fine grained so that there are too many of them, whereas virtual subsystems are too coarse and will be part of every helicopter variant.

At the highest level, helicopter variants are identified by a selection of *features*. Functional features are defined as the largest coherent set of functions that

- are either together included into or excluded from a helicopter variant, or
- are activated or de-activated together, and
- do not change across helicopter variants.

Physical equipment or special capabilities of such equipment implement the functional features. The equipment is represented by virtual devices so that helicopter variants determine the virtual devices to be included into the variant specific software. In ODIN, it is therefore possible to relate devices to the features that they implement – a feature is implemented by at least one virtual device, see Fig. 6.

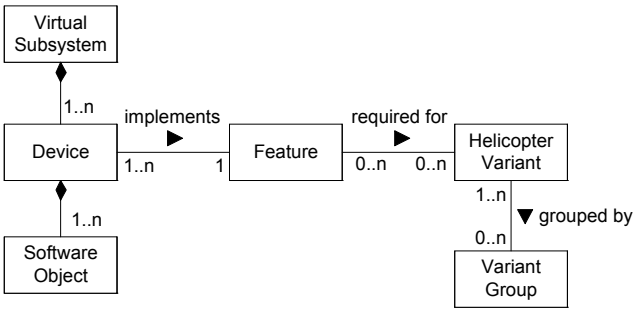

Figure 7: Feature based Code Selection

For variant selection, ODIN must also be able to manage the relation between helicopter variants and the set of features that characterise the variant.

In the NH90 project, there is not a one-to-one correspondence between executable software and helicopter variants. Instead, it is possible that a single executable is used for several variants. The actual helicopter variant is encoded in non-volatile memory and is evaluated at start up of the software [2]. The set of variants that can be supported by a single executable software is defined in a *variant group* (Fig 6).

The extraction script takes a variant group as input and selects all virtual devices that implement at least one feature required for at least one variant in the variant group. This way, the selection of data to be extracted from the database solely depends on the data that are managed in ODIN.

4 Code Generator Qualification

Only a minor part of the total development effort in safety critical military avionics projects is actually induced by coding (see for example [10]). Therefore, full benefit of code generation can only be realised if also verification and documentation activities can be reduced or even eliminated or automated. This approach is only acceptable to certification authorities if the generator tools have been qualified, which is the case for the NH90 code generators. The qualification process was established in a generator qualification plan and agreed with the certification authority.

The generator qualification plan defines a number of test classes, where every test class represents a certain ramification of the extraction or generation logic and hence produces different output. For the code generator components that are the subject of this paper, 56 test classes have been specified. For every test class, there is a

representative test data set defined in ODIN. The code generators take these test data to generate example code.

For every test class, there is also a unit test defined, implemented in a set of test drivers to allow automation of the test execution [11]. The test drivers execute the unit tests on the example code and produce test result documentation automatically.

5 Summary

High level access to low level interface data is a common problem in resource constrained distributed embedded real-time systems. One solution to this problem is to separate the handling of low level interface data from higher level application logic (*separation of concerns*) and to provide a structured, domain related high-level interface to the application software.

The mapping between low-level interface data representation and the domain related high-level data representation should be generated, at least if there is a high number of interface data involved. Generation relieves software developers from error prone and time consuming low level bit manipulation.

In the NH90 software project, code generation is one of the pillars on which the implementation of the NH90 Software Product Line relies. The variations induced by the different system configurations have a large impact on the messages and signals transmitted over the avionics data buses. With code generation, developers do not need to take care about inclusion or exclusion of signals and messages for the different variants.

The code generators in the NH90 project have been qualified in order to reduce the effort required for verification and documentation of generated code, as it is required for safety critical avionics software. The overall reduction in development effort is significant since more than 50 percent of the software of the NH90 avionics main computers is generated.

References

[1] F. Dordowsky and W. Hipp (2009), *Adopting software product line principles to manage software variants in a complex avionics system*, in *Proceedings of the 13th International Software Product Line Conference volume 1*, J. D. McGregor and D. Muthig (eds), Software Engineering Institute.

[2] R. Bridges, F. Dordowsky, and H. Tschöpe (2011), *Implementing a software product line for a complex avionics system in Ada 83*, Ada User Journal 32(2), pp 107–114.

[3] R.Bridges (2007), *NH90 helicopter avionics systems from the 1990s to 2010 and beyond*, in Workshop *Software-Architekturen für Onboardsysteme in der Luft- und Raumfahrt*. Fachausschuss T6.4 Software Engineering, Deutsche Gesellschaft fuer Luft- und Raumfahrt.

[4] RTCA DO-178B Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE Std. ED-12B/DO-178B, December 1992.

[5] G. Budich (2000), *Generation of Ada code from specifications for NH90 computers*., in *Proceedings of the 26th European Rotorcraft Forum*, pp 26–29.

[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal (1996), *Pattern-oriented Software Architecture: A System of Patterns*, John Wiley & Sons.

[7] E. Gamma, R. Helms, R. Johnson, and J. Vlissides (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[8] D. Chamberlain, D. Cross, and A. Wardley (2003), *Perl Template Toolkit*, O'Reilly Media. Used in NH90 for code generation.

[9] F. Dordowsky, R. Bridges, and H. Tschöpe (2011), *Implementing a software product line for a complex avionics system*, in *Proceedings of the 15th International Software Product Line Conference,* I. Schaefer, I. John, and K. Schmid (eds), IEEE Computer Society, pp 241–250.

[10] C. Jones (2002), *Software cost estimation in 2002*, in *CrossTalk: The Journal of Defense Software Engineering*, pp 4–8.

[11] I. Groselj (2006), *Prozeß zur qualifizierung von generiertem code im nh90 sw - projekt*. In *Workshop Software-Architekturen für Onboardsysteme in der Luft- und Raumfahrt.* Fachausschuss T6.4 Software Engineering, Deutsche Gesellschaft fuer Luft- und Raumfahrt.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/adaanswers/gems.

Gem #127: Iterators in Ada 2012 - Part 1

Emmanuel Briot, AdaCore

Abstract. Ada 2012 iterators provide syntactic sugar for iterating over data structures. This Gem describes the new syntax and what it brings to the language. Part 2 will explain how you can define your own iterators when formulating new data structures.

Let's get started…

The following examples assume we have instantiated an Ada list such as:

```
 with Ada.Containers.Doubly_Linked_Lists;
...
declare
   package Integer_Lists is
      new Ada.Containers.Doubly_Linked_Lists (Integer);
   use Integer_Lists;

   L : Integer_Lists.List;

begin
   L.Append (10);
   L.Append (20);
end;
```

In Ada 2005, an iteration over this list would look like:

```
declare
   C : Integer_Lists.Cursor;
begin
   C := First (L);
   while Has_Element (C) loop
      -- Print current value
      Put_Line (Integer'Image (Element (C)));

      -- Change the element in place in the list
      Replace_Element (L, C, Element (C) + 1);

      Next (C);
   end loop;
end;
```

If the list contains elements more complex than integers (controlled types for instance), the above code is not very efficient, since a call to function Element will return a copy of the element. To avoid a copy, one could use a nested subprogram and the procedures Query_Element and Update_Element, but that would make the code more complex and less readable.

Ada 2012 defines three forms of iterators. The first form is called a generalized iterator. The syntax and semantics for it is given in the Ada 2012 Reference Manual (5.5.2), but here is an example of its use:

```
for C in L.Iterate loop
      Put_Line (Integer'Image (Element (C)));
      Replace_Element (L, C, Element (C) + 1);
 end loop;
```

The third form of iterator, called an array component iterator, is similar to a container element iterator, but applies to array types. Here is an example of this form:

```
declare
   Arr : array (1 .. 2) of Integer := (1 => 10, 2 => 20);
begin
   for E of Arr loop
      Put_Line (Integer'Image (E));
      E := E + 1; -- Change in place
   end loop;
end;
```

As the example shows, we can even modify the iterator element E directly, and this modifies the value in the list itself. This is also efficient code when the list contains complex types, since E is not a copy of an element, but a reference to it.

The second part of this Gem series will explain how to write your own iterators and how the loops shown above are expanded by the compiler.

Gem #128: Iterators in Ada 2012 - Part 2

Emmanuel Briot, AdaCore

Abstract: The first part of this two-part Gem series explained the basics of Ada 2012 iterators. Part 2 goes into more detail, showing how to create iterators for user-defined data structures.

Let's get started…

In Part 1, we discussed the basic forms of iterators in Ada 2012 and gave some simple examples. This part goes into greater detail, showing how to create iterators for your own data structures. We'll start by learning about two supporting features introduced in Ada 2012.

The first is the new generic package Ada.Iterator_Interfaces. This package defines two abstract types Forward_Iterator and Reverse_Iterator. The intent is that each container should declare extensions of these and provide concrete implementations for their primitive operations. Briefly, an iterator encapsulates a cursor and a container, and hides the First, Has_Element, and Next operations.

The second new feature is that of a reference type. A reference type is a record with an access discriminant that defines the "Implicit_Dereference" aspect. This is the actual type manipulated by the container element iterators, and the aspect

eliminates the need to write ".all" every time an element is referenced.

Here is an example of such a declaration, taken from the standard package Ada.Containers.Doubly_Linked_Lists:

```
type Constant_Reference_Type
   (Element : not null access constant Element_Type)
     is private with Implicit_Dereference => Element;
```

Whenever we have such a reference, for example E of type Constant_Reference_Type, we can just use the name "E", and this is automatically interpreted as "E.Element.all". Another advantage of this type over a simple access to element is that it ensures the user cannot accidentally free the element.

Now that we understand what iterators and references are, we can start applying them to our own data structures.

Let's assume we are creating our own data structure (such as a graph, a queue, or anything that is not a direct instantiation of an Ada 2005 container). The following examples are framed as a "list", but this really applies to any data structure. Let's also assume that the container holds unconstrained elements of type "T'Class", giving us a more realistic and interesting example than the Part 1 example that just contained Integers.

To provide iterators for this data structure, we need to define a number of Ada 2012 aspects, described in more detail below.

```
type T is tagged null record;  -- any type

type T_List is ...  -- a structure of such types
  with Default_Iterator  => Iterate,
       Iterator_Element  => T'Class,
       Constant_Indexing => Element_Value;

type Cursor is private;
function Has_Element (Pos : Cursor) return Boolean;
-- As for Ada 2005 containers

package List_Iterators is
  new Ada.Iterator_Interfaces (Cursor, Has_Element);

function Iterate (Container : T_List)
  return List_Iterators.Forward_Iterator'Class;
-- Returns our own iterator, which in general will be
-- defined in the private part or the body.

function Element_Value (Container : T_List;
         Pos : Cursor) return T'Class;
-- Could also return a reference type as defined in the
-- Part 1 Gem
```

For those unfamiliar with aspects in Ada 2012, it's worth noting that they can be forward references: in the case above, for instance, the aspect "Default_Iterator" is defined before Iterate is declared (and we could not declare it first in any case, since the function Iterate needs to know about T_List).

To understand the aspects, let's look at how the generalized iterators are expanded by the compiler.

This loop:

```
for C in List.Iterate loop   -- C is a cursor
  declare
    E : T'Class := Element (C);
  begin
    ...
  end;
end loop;
```

is expanded into:

```
declare
  Iter : Forward_Iterator'Class := List.Iterate;
  -- Default_Iterator aspect
  C : Cursor := Iter.First;  -- Primitive operation of iterator
begin
  while Has_Element (C) loop
    -- From Iterator_Interfaces instance
    declare
      E : T'Class := List.Element_Value (C);
      -- Constant_Indexing aspect
    begin
      ...
    end;
    C := Iter.Next (C);   -- Primitive operation of iterator
  end loop;
end;
```

The subprogram Iterate, referenced in the "Default_Iterator" aspect, creates and returns a new iterator. In general, it will also hold a reference to the container itself to ensure the container lives at least as long as the iterator.

The iterator is then used to get and manipulate a cursor. Retrieving an element from the cursor is done via the function defined in the "Constant_Indexing" aspect. (A similar aspect "Variable_Indexing" is used when the loop needs to write the element, but we will not demonstrate that here.)

The function Element_Value is written here in its simplest form: it directly returns a copy of the element contained in the data structure. We could choose instead to return a reference type as explained in the Part 1 Gem, to avoid copies of the elements. (Note that in the case of Variable_Indexing, the function's result type must be a reference type.)

The container element iterators are expanded similarly. The only difference is that the cursor C is not visible.

For the actual implementation of Iterate and Element_Value, we recommend looking at the implementation for the standard containers, such as Doubly_linked_Lists. All of the Ada 2005 containers were enhanced to support iterators, and these provide various examples of code that can be reused for your own applications.

Finally, let's look at a code pattern that might be useful. The test case is the following: we have implemented a complex data structure that contains elements of type T'Class. When we use the container element iterators, E is thus of type T'Class, which we can express with the following syntax:

```
for E : T'Class of List loop
  ...
end loop;
```

Now let's consider a type TChild that extends T. We can still store elements of type TChild in the data structure, but we then need explicit conversions in the loop above to cast E to TChild'Class. We would like to minimize the amount of code needed to create a container that holds TChild'Class elements. For instance:

```
type TChild_List is <see full type below>;
```

```
   Child_List : TChild_List;
   for E of Child_List loop
      -- E is of type TChild'Class, so no conversion is needed.
   end loop;
```

Of course, one possibility is to make our container generic and instantiate it once for T'Class, once for TChild'Class, and so on. That's certainly a minimal amount of Ada source code, but it can still represent a significant amount of compiled code and will increase the size of the final executable. In fact, we can simply mirror the T / TChild hierarchy in the containers themselves and redefine only a minimal number of aspects to achieve the goal.

```
   type TChild_List is new T_List with null record
      with Constant_Indexing => Child_Value,
           Default_Iterator  => Iterate,  -- inherited from T_List
           Iterator_Element  => TChild'Class;

   function Child_Value (Self : TChild_List; Pos :
Cursor'Class);
      return TChild'Class is
   begin
      return TChild'Class (Element_Value (Self, Pos));
   end Child_Value;
```

The amount of additional code is minimal (just one extra function, which is likely to be inlined), and now we can write the container element loop with no need for conversions. Since the containers themselves are now organized as a hierarchy, we can have subprograms that work on a T_List that also work on a TChild_List (the usual reuse of object-oriented code).

However, the new structure is not perfect. One caveat is that it's possible to insert an object of type T in a TChild_List (because the list contains T'Class elements). The consequence is that the iterator will raise Constraint_Error in the implicit call to Child_Value in the expanded code.

We hope that this Gem has helped to explain some of the "magic" behind the Ada 2012 iterators and containers, and will enable you to use them more effectively in your own code. Even though they do require quite a lot of boilerplate code, written once up front for a container, they definitely make code in clients of the container easier to read and understand.

One final note: the examples in this Gem require a fairly recent version of the compiler, which includes a number of adjustments to reflect recent clarifications in the Ada 2012 rules.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o K.U. Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada in Sweden

Ada-Sweden
attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada Switzerland

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch