ADA USER JOURNAL

Volume 33
Number 2
June 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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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 June issue of the Ada User Journal was finalized after the successful Ada-Europe conference, which took place in Stockholm this June. I would like to highlight the enthusiastic mood of the conference, which was felt both on the technical (and non-technical) sessions and on the networking environment during the breaks, lunches and banquet. It was really enjoyable, and worthwhile the effort. An effort that also caused some delays in producing this issue of the Journal; we will get back to the regular publication schedule already in the next issue.

Concerning contents, this issue starts as usual with the News Digest (the first solo issue of the new News Editor, Jacob Sparre Andersen), Calendar and Forthcoming Events sections. In the latter, the reader will find the advance information of the 2012 SIGAda High Integrity Language Technology conference, and the preliminary call for papers for the 18th International Conference on Reliable Software Technologies – Ada-Europe 2013 that will take place next year in Berlin, Germany. We also print the joint press release of the Ada Resource Association (ARA) and Ada-Europe, on the Ada 2012 language standard being submitted to ISO.

The technical contents of the Journal start with another installment of the Ada 2012 Rationale, with the chapter on the changes on structure and visibility arising in the new standard. Probably the most noticeable change is that functions may now have parameters of all modes, but the chapter also describes the changes to incomplete types, use clauses and extended return statements.

Afterwards we start publishing the Proceeding of the Industrial Track of Ada-Europe 2012, with a paper by Steen Palm, from Terma A/S, Denmark, describing the approach used in the development of software for instruments on a payload of the International Space Station. Continuing with material derived from the conference, the issue also provides extended abstracts of two tutorials on SPARK, by Trevor J. Jennings, from Altran-Praxis, UK.

Finally, we conclude with the usual Ada Gems section, with one gem by Thomas Quinot, of AdaCore, with a new installment of the Ada DSA Gems series, and two gems by Christoph Grein, of Ada Magic, presenting some advantages that Ada 2005 and Ada 2012 bring to references in containers.

*Luís Miguel Pinho*
*Porto*
*June 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

### The AdaIC web site

*From: Randy Brukardt*
*Date: Wed, 16 May 2012 18:14:17 -0500*
*Subject: AdaIC news: news submission*
*Newsgroups: comp.lang.ada*

The AdaIC news feed is going to be revamped in response to feedback that we've received (including some from here). As part of that, we've established a new e-mail address: news@adaic.org

Any Ada-related news items will be eligible, whether they are from community projects or from commercial vendors, and whether they're from ARA members or non-members. (The posting criteria will be different for members and for non-members, of course; recall that any organization [it doesn't have to be commercial] can join the ARA.)

In particular, we encourage those of you who post announcements here to also send those announcements to the news@adaic.org e-mail address. (If you're currently sending announcements to the webmaster address, you don't have to change, but please send the announcements to only one or the other address, since they both go to the same people.)

*From: Randy Brukardt*
*Date: Wed, 16 May 2012 18:17:27 -0500*
*Subject: AdaIC news: short user survey*
*Newsgroups: comp.lang.ada*

As part of our ongoing efforts to improve the AdaIC website, we're conducting a short survey of Ada users to guide possible improvements to the site. Find the survey at:
http://www.adaic.org/survey/

P.S. You'll need to have Javascript enabled to access the survey, as I found out when I tried it. :-)

## 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]

### Video of FOSDEM keynote by Robert Dewar

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Wed, 14 Mar 2012 06:09:57 -0700*
*Subject: Robert Dewar at FOSDEM 2012: video available*
*Newsgroups: comp.lang.ada*

In case you didn't notice, Robert Dewar's keynote speech has been available in video for a month thanks to the outstanding work of the FOSDEM video team:

http://video.fosdem.org/2012/maintracks/janson/Free_Software__A_Viable_Model_for_Commercial_Success.webm

Enjoy!

In case you ask: no, the presentations in the Ada developers' room were not filmed.

*From: roderick.chapman@googlemail.com*
*Date: Wed, 14 Mar 2012 12:40:08 -0700*
*Subject: Re: Robert Dewar at FOSDEM 2012: video available*
*Newsgroups: comp.lang.ada*

[…]

A small factual correction to Robert's talk: Praxis is part of the Altran Group, not Alcatel...

### Open Ada-DK Meetings

*From: Ada in Denmark*
*Date: Mon, 12 Mar 2012 14:40:35 +0000*
*Subject: Ada in Denmark: The March 2012 Open Ada-DK Meeting*
*URL: http://ada-dk.org/2012/03/the-march-2012-open-ada-dk-meeting/*

March 13th. 2012 from 1730 -> ? marks the day and time when the sixteenth 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.

If you’re interested in participating, feel free to send us an email and we’ll inform you of the when and where. You can also ping me at Google+ (https://plus.google.com/u/0/112815721307813813920/posts), identi.ca (http://identi.ca/thomaslocke) or join the Freenode IRC #ada channel and look for ThomasLocke.

The meeting is of course free.

[Also open Ada-DK meetings April 3rd, May 8th and June 5th. —sparre]

### Ada-Belgium Spring 2012 Event

*From: dirk@vana.cs.kuleuven.be. (Dirk Craeynest)*
*Date: Sun, 6 May 2012 22:20:36 +0000*
*Subject: Ada-Belgium Spring 2012 Event incl. Ada workshop*
*Newsgroups: comp.lang.ada,fr.comp.lang.ada,be.comp.os.linux,be.comp.programming*

---

Ada-Belgium Spring 2012 Event

Saturday, May 12, 2012, 12:00-19:00

Wavre area, south of Brussels, Belgium

including at 15:00

2012 Ada-Belgium General Assembly

and at 15:45

Ada Workshop

<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/local.html>

---

Announcement

---

The next Ada-Belgium event will take place on Saturday, May 12, 2012 in the Wavre area, south of Brussels.

For the fifth year in a row, Ada-Belgium decided to organize their "Spring Event", which starts at noon, runs until 7pm, and includes an informal barbecue, a key signing party, the 19th General Assembly of the organization, and a two-part workshop consisting of a recorded presentation by Robert Dewar on Ada, and practical hands-on experience on packaging Ada software for Debian with Ludovic Brenta, principal maintainer of Ada in Debian.

Schedule

---

- 12:00 welcome and getting started (please be there!)
- 12:15 informal barbecue
- 14:45 key signing party
- 15:00 Ada-Belgium General Assembly
- 15:45 recorded Ada presentation by Robert Dewar
- 16:30 workshop on creating Debian packages of Ada software
- 19:00 end

[…]

### Ada Europe 2012 (a quote)

*Subject: Notices tagged with ae2012*
*URL: http://identi.ca/tag/ae2012*

[Software] “Engineers _serve_ other people” - Göran Backlund

## Ada-related Resources

### AdaCommons website moved

*From: Tero Koskinen <tero.koskinen@iki.fi>*
*Date: Wed, 28 Mar 2012 21:19:57 +0300*
*Subject: AdaCommons (Re: AdaRocks.com)*
*Newsgroups: comp.lang.ada*

[…]

The new address of AdaCommons is http://commons.ada.cx/

### Currently maintained Ada libraries

*From: i3text@gmail.com*
*Date: Thu, 14 Jun 2012 20:51:31 -0700*
*Subject: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

When searching for Ada libraries, it is discouraging to find mostly dead links or links to dead projects. So I decided to make a list of all the libraries I could find that seem to be still active projects. To be listed, a library must have seen some maintenance activity within the last year. I generally ingored things that are already part of the GNAT distribution.

At first, I wanted to list just libraries of production quality, but I figured I couldn't reliably evaluate that. So, a lot of the things here are alpha quality. Of course, there are libraries that haven't seen any work for several years but still get downloaded a lot. But I wanted a listing of libraries where I could reasonably expect help and bug-fixes.

Do you know any I've missed?

---

Ada 2005 Math Extensions -- Additions to the GNAT libraries.

http://sourceforge.net/projects/gnat-math-extn/

Ada 95 Booch Components -- Container library.

http://sourceforge.net/projects/booch95/

ada-ado -- Mapping for object-relational databases.

http://code.google.com/p/ada-ado/

ada-asf -- Ada Server Faces.

http://code.google.com/p/ada-asf/

ada-awa -- Ada Web Application.

http://code.google.com/p/ada-awa/

adabindinggmpmpfr -- Binding to the GNU GMP and MPFR.

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

Ada Class Library -- Scripting tools.

http://sourceforge.net/projects/adacl/

Ada Cryptographic Objects -- Library of crypto primitives.

http://www.assembla.com/code/acrypto/subversion/nodes

ada-el -- A expression languge like JSP.

http://code.google.com/p/ada-el/

ada-gen -- Dynamo, a aid for web applications.

http://code.google.com/p/ada-gen/

ada-security -- Ada web security framework.

http://code.google.com/p/ada-security/

Ada Spawn Manager -- Control for spawing processes.

http://www.codelabs.ch/spawn-manager/index.html

ada-util -- Ada Utility Library.

http://code.google.com/p/ada-util/

Ahven -- A simple unit testing library for Ada 95.

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

aicwl -- Industrial control GUI widgets.

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

ARM-Ada -- Some libraries for compiling to ARM.

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

Basil -- MIME library for Ada 2005.

http://hafdconsulting.com/projects/basil/

components -- Misc objects.

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

Config -- A package for parsing configuration files.

http://sourceforge.net/projects/ini-files/

Debug -- Trace support for multitasking programs.

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

Deepend -- A storage pool with subpool capabilities for Ada 2005.

http://sourceforge.net/projects/deepend/

dequesterity -- Deque/buffer generics in Ada 2005.

http://sourceforge.net/projects/dequesterity/

EWS -- Embedded Web Server.

http://sourceforge.net/projects/embed-web-srvr/

Excel_Out -- A package for writing Excel files.

http://sourceforge.net/projects/excel-writer/

Florist -- Ada POSIX binding.

http://sourceforge.net/projects/gnat-florist/

fuzzy -- Fuzzy sets for Ada 2005.

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

Generic Image Decoder -- Package for image decoding.

http://sourceforge.net/projects/gen-img-dec/

GLOBE_3D -- Real-time 3D engine based on OpenGL.

http://sourceforge.net/projects/globe3d/

GNADE -- GNat Ada 95 Database Environment.

http://sourceforge.net/projects/gnade/

gtkada_contributions -- Extras for GtkAda.

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

GWindows -- GUI framework for Windows.

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

intervals -- Interval arithmetic library.

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

KOW -- Framework for database management.

http://framework.kow.com.br/

libadanix -- A POSIX binding for GNAT.

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

libredave -- Library for using Simatic S5/S7.

http://sourceforge.net/projects/librdave/

libsparkcrypto -- A verified cryptographic library in SPARK.

http://senier.net/libsparkcrypto/

match -- Pattern matching library.

http://www.dmitry-kazakov.de/match/match.htm

Mathpaqs -- Various math packages.

http://sourceforge.net/projects/mathpaqs/

Matreshka -- Text, database, and web framework.

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

ORBit-Ada -- Binding to the ORBit Corba ORB.

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

OS_Services -- Access to common operating system features.

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

paraffin -- Ada 2005 generics to support parallelism.

http://sourceforge.net/projects/paraffin/

PCSC/Ada -- Bindings to PC/SC middleware.

http://www.codelabs.ch/pcscada/

player-ada -- Bindings for the player robotic platform.

https://github.com/mosteo/player-ada

plplot -- Bindings to the PLplot plotting library.

http://plplot.sourceforge.net/

PNG_IO -- Ada 95 coder/decoder for PNG files.

http://sourceforge.net/projects/png-io/

portaudioada -- Binding to PortAudio.

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

Protection -- Protection for semaphores and procedures.

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

QtAda -- Bindings to the Qt GUI framework.

http://www.qtada.com/

SOCI-Ada -- Wrapper for the SOCI database library.

http://www.inspirel.com/soci-ada/

Storage_Stream -- Write data of any type at any location.

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

strings_edit -- Strings library with UTF-8 and pattern matching.

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

tables -- Search in tables with string keys.

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

TclAdaShell -- Binding to Tcl/Tk.

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

Templates_Parser -- Library for generating HTML.

http://www.obry.net/

units -- Library for handling units of measurement.

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

v8a -- Binding to V8 JavaScript engine.

http://code.google.com/p/v8a/

Variable_String -- Support for variable length strings.

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

VTKAda -- Ada 2012 bindings to Visualization Toolkit.

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

[…]

YAMI4 -- Messaging library for distributed systems.

http://www.inspirel.com/yami4/

[…]

ZanyBlue -- A framework for finite element analysis.

http://sourceforge.net/projects/zanyblue/

Zip-Ada -- A library for .zip archives.

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

*From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>*
*Date: Thu, 14 Jun 2012 22:37:59 -0700*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

[…]

Your criteria are somewhat strange. A mature library will be useful but see no "maintenance activity" because there is no longer anything to change. For example, the PragmAda Reusable Components are very much alive and supported, but have not been changed recently:

http://pragmada.x10hosting.com/

*From: i3text@gmail.com*
*Date: Fri, 15 Jun 2012 13:04:02 -0700*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

[…]

This is actually the first time I've seen someone take this position, so I've been thinking about when it might be true.

One the one hand, I agree that this list is unsatisfactory as a collection of all libraries that one might want to use. An old but well crafted library could still be valuable, especially if it covers something important and there is no alternative.

However, my "activity" criterion was chosen for two reasons. First, I take it as axiomatic that no software achieves perfection. If it is being used, people will report bugs and rough edges, prompting the developer to tweak things, even if it is just the documentation. Second, a large portion of libraries interact with other things that continue to evolve, so compatibility at one point doesn't imply compatibility forever.

I have not done embedded systems work, but I've heard that it is easier than doing desktop applications since you are developing for a restricted, stable environment. In the field of embedded systems, is it a common attitude that software *can- be developed to a point where it no longer needs any attention? I'm wondering how this might affect how Ada is received by application developers.

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Fri, 15 Jun 2012 17:41:04 -0400*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

> This is actually the first time I've seen someone take this position, so I've been thinking about when it might be true.

That's kind of a sad commentary on the state of the art, isn't it? If you're not constantly fixing bugs (and adding new ones?) people consider your project "dead".

[…]

*From: "Nasser M. Abbasi" <nma@12000.org>*
*Date: Fri, 15 Jun 2012 17:58:25 -0500*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

[…]

I do not believe that a software library can ever not go continuous maintenance. There is always more testing to be done, more features to add, make the software more robust, test against new changes outside of the control of the library.

Software that does not change, means it is not used much. Software that is used by many, means there is always new functionality needed, enhancement requests, bug fixes, test cases added, etc...

Software for me is a living thing. Stop taking care of it, stop feeding it, stop growing it, it will die soon.

If it does not die on its own, something better will come along, with better features, and replace it.

*From: "J-P. Rosen" <rosen@adalog.fr>*
*Date: Sat, 16 Jun 2012 08:11:04 +0200*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

[…]

Counter-example: my "debug" package.

I didn't change anything for years. It does its job, full stop.

*From: Vadim Godunko <vgodunko@gmail.com>*
*Date: Fri, 15 Jun 2012 23:35:46 -0700*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

AXMPP - implementation of XMPP protocol

http://forge.ada-ru.org/axmpp

*From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>*
*Date: Fri, 15 Jun 2012 23:47:25 -0700*

*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

[…]

Right. I know of projects using the PragmARCs. There are probably others I don't know of. But there hasn't been an error report or enhancement request for years.

*From: Stephen Leake <stephen_leake@stephe-leake.org>*
*Date: Sat, 16 Jun 2012 09:24:52 -0400*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

> When searching for Ada libraries, it is discouraging to find mostly dead links or links to dead projects. So I decided to make a list of all the libraries I could find that seem to be still active projects. To be listed, a library must have seen some maintenance activity within the last year.

That's a little short, but you do need some cutoff.

A related criteria would be postings on a dedicated mailing list, or here.

> Do you know any I've missed?

mine: http://stephe-leake.org/

But I understand why; I haven't posted a new version of SAL since 2009! It really is time to update; I use these libraries all the time at work, and there is new stuff. I get very little feedback from users, so I tend to forget about the website.

*From: "RasikaSrinivasan@gmail.com" <RasikaSrinivasan@gmail.com>*
*Date: Sat, 16 Jun 2012 07:04:27 -0700 (PDT)*
*Subject: Re: Listing currently maintained libraries for Ada*
*Newsgroups: comp.lang.ada*

Not for production, but for learning:

Mine: projectlets.sourceforge.net

## Ada Crypto Library

*Author: Christian Forler <cforler@gmx.de>*
*Date: Tuesday, 12-Jun-12 16:03:34*
*Subject: Welcome to the Ada Crypto Library (ACL) aka libadacrypt-0.5.x*
*URL: https://github.com/cforler/Ada-Crypto-Library*

This library is an ongoing project with the goal to provide "strong" and clean coded cryptography library for Ada.

[…]

Implemented Features:

- Symmetric Cryptography
  - o Blockciphers: AES, Twofish, 3DES, Serpent
  - o Modes of Operation : BPS, CFB, Ctr, OFB
  - o Tweakable Blockcipher Modes: CMT, XT
  - o Hash functions: SHA-1 (broken), SHA-256, (SHA-384), SHA-512, Whirlpool
  - o MACs: RMAC, HMAC, CMAC
  - o Authenticated Encryption schemes: OCB, SIV and McOE
- Big (unsigned) number library
  - o Primary cyclic group arithmetic (Z_p)
  - o Binary Field arithmetic support.
  - o Elliptic Curve arithmetic
  - o Supersingular Elliptic Curves Over Binary Fields (SS-BF)
  - o Non-Supersingular Elliptic Curves Over Binary Fields (NSS-BF)
  - o Elliptic Curves Over Z_p (EC-Z_P)
- Asymmetric Cryptography
  - o Probabilistic primality testing
  - o DSA signature scheme
  - o OEAP-RSA
  - o ECDSA, ECDH
- Nonce Generator Support: Random, Counter, Mixed
- AUnit-3.4 based Test suite

The library has been tested on the following platforms:

- Linux debian 3.1.0-1-amd64 (i5 CPU)

[…]

## Ada Database Objects

*From: Stephane Carrez*
*Date: May 12 2012, 22:17*
*Subject: Ada Database Objects 0.3.0 is available*
*URL: http://blog.vacs.fr/index.php?post/2012/05/12/Ada-Database-Objects-0.3.0-is-available*

The Ada Database Objects is an Object Relational Mapping for the Ada 2005 programming language. It allows to map database objects into Ada records and access databases easily. Most of the concepts developed for ADO come from the Java Hibernate ORM. ADO supports MySQL and SQLite databases.

The new version brings:

- Support to update database records when a field is really modified,
- Customization of the SQLite database connection by using SQLite PRAGMAs,
- Escape of MySQL or SQLite reserved keywords,
- Support for blob type.

This version can be downloaded at

http://code.google.com/p/ada-ado/downloads/list

## Industrial Control Widget Library

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sun, 13 May 2012 21:16:43 +0200*
*Subject: ANN: Ada industrial control widget library v1.3*
*Newsgroups: comp.lang.ada*

The library is based on GtkAda and cairoada, Ada bindings to GTK+ and cairo. The key features of the library:

- Widgets composed of transparent layers drawn by cairo;
- Fully scalable graphics;
- Support of time controlled refresh policy for real-time and heavy-duty applications;
- Caching graphical operations;
- Stream I/O support for serialization and deserialization;
- Ready-to-use gauge, meter, oscilloscope widgets;
- Editor widget for WYSIWYG design of complex dashboards.

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

[…]

## Ada Server Faces

*From: Stephane Carrez <Stephane.Carrez@gmail.com>*
*Date: Fri, 11 May 2012 23:38:24 +0200*
*Subject: [Ann]: Ada Server Faces 0.4.0 is available*
*Newsgroups: comp.lang.ada*

Ada Server Faces is a web framework which uses the Java Server Faces design patterns.

This new version brings a serious step ahead towards JSF compatibility. This new version provides:

- Support for shared or static build configuration,
- Support for file upload,
- New components <h:inputFile>, <f:metadata>, <f:viewParam>, <f:viewAction>,
- New EL function util:hasMessage,
- ASF now Implements the JSF phase events and phase listeners,
- Implements the JSF/Ruby on Rails flash context,
- Adds the pre-defined JSF beans: initParam, flash,
- Support for locales and honors the Accept-Language,
- New demos are available in French and English

It has been compiled and ported on Linux, Windows and Netbsd (gcc 4.4, GNAT 2011, gcc 4.6.2). You can download this

new version at http://code.google.com/p/ada-asf/downloads/list.

A live demo is available at: http://demo.vacs.fr.

Feel free to play with the OpenID stuff!!!

## Ada Utility Library

*From: Stephane Carrez <Stephane.Carrez@gmail.com>*
*Date: Thu, 10 May 2012 08:03:36 +0200*
*Subject: [Ann]: Ada Utility Library 1.5.0 is available*
*Newsgroups: comp.lang.ada*

Ada Utility Library is a collection of utility packages for Ada 2005. A new version is available which provides:

- Concurrent fifo queues and arrays
- Changed Objects.Maps to use a String instead of an Unbounded_String as the key
- Support for shared or static build configuration
- Implementation of input/output/error redirection to a file for process launch

It has been compiled and ported on Linux, Windows and Netbsd (gcc 4.4, GNAT 2011, gcc 4.6.2). You can download this new version at http://code.google.com/p/ada-util/downloads/list.

## Paraffin

*From: Brad Moore <brad.moore@shaw.ca>*
*Date: Wed, 14 Mar 2012 21:44:17 -0600*
*Subject: ANN: Paraffin 3.2*
*Newsgroups: comp.lang.ada*

I am pleased to announce the availability of Paraffin 3.2

I know it's only been a few days since the 3.1 release, but I applied the same technique that improved iterative work stealing to iterative work seeking, and found some somewhat improved results, plus a bigger improvement in ease of use. More details below.

Paraffin is a set of Ada 2005 generics that may be used to add parallelism to iterative loops and recursive code.

Paraffin also includes a suit of useful parallel utilities that utilize the Paraffin generics. These include generics for;

- integrating a function in parallel
- applying quicksort algorithm in parallel to an array
- applying fast fourier transform to an array of data.
- Red-Black tree container that performs some operations in parallel.
- function to solve matrices using Gauss-Jordan Elimination

The latest stable release and older releases may be downloaded from:

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

For those who want the current development versions of the source they can download using git (http://git-scm.com/) by issuing the following commands;

```
mkdir sandbox
cd sandbox
git clone git://paraffin.git.sourceforge.net/gitroot/paraffin/paraffin
```

[…]

## OAuth 2.0 and Ada Programming

*From: Ada in Denmark*
*Date: Wed, 06 Jun 2012 07:55:31 +0000*
*Subject: OAuth 2.0 and Ada Programming*
*URL: http://ada-dk.org/2012/06/oauth-2-0-and-ada-programming/*

So you need to flex the power of the OAuth 2.0 framework and and you want to use Ada to get the job done, […]

Well today is your lucky day then, as the ever productive Stephane Carrez just churned out an article about OAuth 2.0 and Ada:

http://blog.vacs.fr/index.php?post/2012/06/03/Using-the-Facebook-API

"Through this article you will learn how to use the OAuth 2.0 framework to let an application access service provider APIs such as Facebook API, Google+ API and others. Althought this article uses Ada as programming language and Facebook as service provider, most part also applies to other programming languages and other service providers."

The article goes into great detail, so it's a pretty good read even if you just want to understand how OAuth 2.0 works. If you already know all about OAuth 2.0 and just want to get cracking on some code, how about taking a closer look at Stephane's Ada Security package?

http://code.google.com/p/ada-security/

## Strings_Edit library

*From: Dmitry A. Kazakov*
*Date: Wed, 4 Apr 2012 13:49:30 +0200*
*Subject: ANN: Strings_Edit 2.8*
*Newsgroups: comp.lang.ada*

The library provides means for formatting and editing strings:

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

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

The new version 2.8 includes the package Strings_Edit.Lexicographical_Order for lexicographically comparisons of strings containing chains of digits treated as decimal numbers. These can be used to compare and order file names like abc-123.txt, e.g. for creating sets of such strings etc.

## GtkAda contributions

*From: Dmitry A. Kazakov*
*Date: Sun, 13 May 2012 16:41:00 +0200*
*Subject: ANN: GtkAda contributions v2.13*
*Newsgroups: comp.lang.ada*

A contribution to GtkAda dealing with the following issues:

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

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

*From: Patrick <patrick@spellingbeewinnars.org>*
*Date: Sun, 13 May 2012 08:38:52 -0700*
*Subject: Re: ANN: GtkAda contributions v2.13*
*Newsgroups: comp.lang.ada*

I have been playing with gtk ada but I was discouraged when I read about tasking in the documentation. It seemed like it was not fully supported. Could you give me some feedback on tasking support in gtk ada? could you tell me more about your tasking contributions?

*From: Dmitry A. Kazakov*
*Date: Tue, 15 May 2012 09:00:46 +0200*
*Subject: Re: ANN: GtkAda contributions v2.13*
*Newsgroups: comp.lang.ada*

[…]

GTK is not thread-safe, that means if you would call any Gtk (and thus GtkAda) subprogram form the thread different from the main thread (more accurately, from the thread running the GTK messages loop), it will most certainly crash.

[…]

GtkAda contributions provide means for an Ada task to engage a rendezvous with the main task. It is safe to use any GTK operations from there. It could be done either in the traditional OO way by overriding a primitive "visitor" operation, or per a generic package instantiation and passing a "service" callback.

GtkAda contributions also provide convenience operations for opening simple message boxes from any Ada task.

Further GtkAda contributions provide task-safe tracing facilities for debugging GtkAda programs. Which supports call stack tracing, visual navigation of the call stack, breaking upon GTK errors and warnings, jumping to the source location using the GPS.

For further information read this:

http://www.dmitry-kazakov.de/ada/gtkada_contributions.htm#1

The subdirectory test_gtkada contains test_gtk_tasking.adb which illustrates usage of tasking support.

## 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

## Fuzzy Sets

*From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>*
*Date: Mon, 9 Apr 2012 10:41:52 +0200*
*Subject: ANN: Fuzzy sets for Ada v5.6*
*Newsgroups: comp.lang.ada*

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

- Confidence factors with the operations not, and, or, xor, +, *;
- Classical fuzzy sets with the set-theoretic operations and the operations of the possibility theory;
- Intuitionistic fuzzy sets with the operations on them;
- Fuzzy logic based on the intuitionistic fuzzy sets and the possibility theory;
- Fuzzy numbers both integer and floating-point ones with conventional arithmetical operations;
- Dimensioned fuzzy numbers;
- Fuzzy linguistic variables and sets of linguistic variables with operations on them;
- Dimensioned fuzzy linguistic variables and sets;
- String-oriented I/O is supported;
- GUI interface based on GTK+ (The GIMP Toolkit) with fuzzy set editors, truth values widgets and renderers, linguistic variables sets editors.

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

## Simple Components for Ada

*From: "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de>*
*Date: Mon, 28 May 2012 12:03:15 +0200*
*Subject: ANN: Simple components for Ada 3.17*
*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

[…]

## VTKAda

*From: Leonid Dulman <leonid.dulman@gmail.com>*
*Date: Sun, 27 May 2012 02:08:40 -0700*
*Subject: I'm pleased to announce VTKAda version 5.10 free edition release 01/06/2012*
*Newsgroups: comp.lang.ada*

I'm pleased to announce VTKAda version 5.10 free edition release 01/06/2012.

VTKAda is Ada-2012 port to VTK (Visualization Toolkit by Kitware, Inc) and Qt4 application and UI framework by Nokia ParaView 3.14.0 (VTK version 5.10.0), Qt version 4.8.1 open source and qt4c.dll(libqt4c.so) were builded with Microsoft Visual Studio 2010 in Windows and gcc in Linux x86.

Package was tested with gnat gpl 2011 ada compiler (-gnat12 option) in Windows XP Sp3 32bit, Windows 7 Sp1 64bit and Debian 5 x86.

VTKAda is powerful 2D and 3D rendering and imaging system and works inside Qt4 application.

Current state of VTKAda is 39549 procedures and function distributed in 634 packages. 134 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 to Qt4 framework and can be used as independent system.

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

I have a dream: to take a part in CNNC nuclear reactor project. I think, it's a great project of 21TH centure and visualization is a big part of this project, but I can not connect them.

Sources and prebuild VTK 5.10.0 and Qt 4.8.1 for win32 and x86 , you can found on my DVD "ADASTUDIO 2012" ,if you are interested to get this DVD tell me know

## Dequesterity

*From: Brad Moore <brad.moore@shaw.ca>*
*Date: Sat, 07 Apr 2012 23:22:21 -0600*
*Subject: ANN: Dequesterity v1.2*
*Newsgroups: comp.lang.ada*

I am pleased to announce the release of version 1.2 of Dequesterity.

Dequesterity is a set of Ada 2005 generics that provide various forms of general purpose buffer containers. Buffers are circular data structures and may be used as deques, queues, ring buffers, stacks, double ended stacks, vectors, and similar abstractions.

Simpler buffer forms are intended for use non-concurrently, while higher level forms provide concurrency support.

Some of the buffers are stream buffers that allow heterogeneous objects to be stored in the buffer. There are also Ravenscar Stream buffers, that allow a writer and a reader task to safely stream heterogeneous objects to the same buffer.

Buffers may be accessed remotely, and can be persistent. There are also indefinite buffers that can store variable sized objects such as strings.

There currently exist 40 generic buffers types, 10 stream buffer types, and 25 preinstantiated string buffer types.

The most significant changes since the previous release are;

- Saving/Loading buffers from files, and Persistent Buffers were not working under Linux. The low level Preelaborated IO had some portability issues in the area of determining the size of a file. The Linux test executable, test_buffers now works as it should.
- Prelaborated_IO Size now returns a long integer instead of Integer. (Which would be 64 bits on 64 bit OS's) This allows buffers to work with much larger files, if buffer content is to be stored persistently.
- Only one project file needed for an executable, that builds for both Windows and Linux
- Cleaned up compiler warnings
- Created a Buffer Demo executable, which is stripped down to demonstrate some buffer concepts.
- Ravenscar example exits after 30 seconds, instead of executing indefinitely.
- Persistent Stream Buffers do not need to be instantiated, similar to the other Stream Buffer types.
- Fixed bug in Unbounded Buffer, if vector to read into is a zero element array, the Read routine does not generate a constraint error.
- Fixed bug in indefinite Unbounded buffers. When finalizing the buffer if the internal buffer is null, don't attempt to dereference the null value.

The latest stable release and older releases may be downloaded from;

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

## Ada Web Application Framework

*From: Stephane Carrez <Stephane.Carrez@gmail.com>*
*Date: Thu, 24 May 2012 23:01:55 +0200*
*Subject: [Ann] Ada Web Application 0.2.0 is available*
*Newsgroups: comp.lang.ada*

Ada Web Application is a framework to build web applications easily on top of Ada Server Faces, Ada Database Objects and Ada Web Server.

The new version of the framework provides:

- A new event framework with configurable action listeners,
- Persistent event queues for the event framework,
- A new blog module and wiki engine supporting Google Wiki, Creole, MediaWiki, phpPP and Dotclear syntax,
- New mail UI components allowing to generate and send email easily with the ASF presentation pages,
- A new Javascript plugin Markedit with jQuery Markedit (MIT License)

This new version can be downloaded at http://code.google.com/p/ada-awa/downloads/list (downloading the awa-all package is recommended to get the project and its dependencies).

A demo of an AWA application is available at http://demo.vacs.fr/atlas/

You may also look at a short video at http://youtu.be/2VOZ4_p7h2o to learn building an application in 5 minutes.

## ZanyBlue localization library

*From: Michael Rohan <michael@zanyblue.com>*
*Date: Sun, 29 Apr 2012 23:20:25 -0700*
*Subject: ANN: ZanyBlue v1.0.0 Beta Available*
*Newsgroups: comp.lang.ada*

After quite a while, a new release of ZanyBlue is now available. This is an Ada library currently targeting localization support for Ada (along the lines of Java properties) with supporting message formatting (with built-in localization for about 20 locales).

The major changes from the initial release are

- Message accessors to make using messages safer (the Ada compiler can check for argument types and number).
- Change of licensing from GPLv2 to simple BSD.
- A simple regression testing application for command line utilities.
- A parameter storage package (internally used for command line and for parameter storage during regression tests).

Please see the the project page on Source Forge for download links, documentation, etc,

http://zanyblue.sourceforge.net

## Turbo Pascal 7 library

*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*

Hello, here is Turbo Pascal 7 library port in Ada.

The very first aim was to provide a help for porting Turbo Pascal programs in Ada. Thus it can be combined with P2Ada translator. http://sourceforge.net/projects/p2ada/

The version 1 was based on Mac bindings Carbon-Ada. http://www.macada.org/macada/Downloads.html

The version 2 is based on graphic library GTK-Ada GPL 2011, then it is multi-plateform ;-) See http://libre.adacore.com/tools/gtkada/

More over 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:

```
1     with TP7.System;
2     with TP7.Crt; - - if you comment this
            line then I/O use stdinout
3
4     procedure Hello_GTKAda is
5       use TP7, TP7.System;
6       N : Byte;
7
8     begin
9       Write ("How many hello ? ");
10      Readln (N);
11      for I in 1 .. N loop
12        Writeln ("Hello with GTKAda
                              console.");
13      end loop;
14    end Hello_GTKAda;
```

See: http://blady.pagesperso-orange.fr/telechargements/tp-ada/tp7ada-mini.png

The complete code is here: http://p2ada.svn.sourceforge.net/viewvc/p2ada/extras/tp7ada/current/

[…]

See screen copies on: http://blady.pagesperso-orange.fr/tp7ada.html

See also (in French): http://blady.pagesperso-orange.fr/creations.html#ada_tp7

All TP7 features are not completely functional, see current status: http://p2ada.svn.sourceforge.net/viewvc/p2ada/extras/tp7ada/current/TurboPascal7.0-Ada.html

All Pascal source codes were translated in Ada with P2Ada translator:

http://sourceforge.net/projects/p2ada/

## Game Programming

*From: Patrick <patrick@spellingbeewinnars.org>*
*Date: Tue, 3 Apr 2012 09:57:05 -0700*
*Subject: Ada game programming or other sound solutions?*
*Newsgroups: comp.lang.ada*

I am still having trouble getting my project started. Sorry for cluttering the list with more dumb questions.

I have abandoned my hopes of creating a gstreamer binding and I am trying to focus on smaller simpler solutions. I basically need to draw pictures to a screen and have keys create sounds as a child types, low latency is import to me.

I was thinking about using the ada-gtk binding and writing a small binding to a sound library. I have been able to to create some small bindings that compiled but I have hit several dead ends as the underlying Linux sound system seem like a murky world. For instances I was able to create a thin binding for both pulse-audio and portaudio but then I could not figure out how to write a program as there is very little documentation even in C.

So my question is, is there anyone building open source games in Ada that use sound? Could someone refer me to some sample code?

If anyone has any general feedback on creating sounds with Ada that would be great too. I have read the chapter in big book of Ada linux programming

*From: Gautier write-only <gautier_niouzes@hotmail.com>*
*Date: Tue, 3 Apr 2012 11:09:48 -0700*
*Subject: Re: Ada game programming or other sound solutions?*
*Newsgroups: comp.lang.ada*

About sounds/music: did you consider OpenAL [1] ?

There is or was a binding (OpenALAda).

For the graphics you might consider OpenGL. There are Ada-aware bindings and a 3D framework in the GLOBE_3D project [2]. IIRC OpenGL is supported by GTK.

[…]

[1] http://en.wikipedia.org/wiki/OpenAL

[2] http://globe3d.sf.net

*From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>*
*Date: Tue, 03 Apr 2012 13:46:23 -0700*
*Subject: Re: ada game programming or other sound solutions?*
*Newsgroups: comp.lang.ada*

[…]

In simple cases under Linux, spawning a process that executes ogg123 will play an Ogg Vorbis file. Latency could be an issue with this approach.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Sat, 07 Apr 2012 13:12:20 +0200*
*Subject: Re: Ada game programming or other sound solutions?*
*Newsgroups: comp.lang.ada*

[…]

See a largish example of how to use AdaSDL here:

http://www.ada-france.org:8081/branch/changes/org.ludovic-brenta.defendguin

I wrote that to teach myself Ada and SDL a few years ago :)

# Ada-related Tools

## Ahven Unit Test Framework

*From: Ada in Denmark*
*Date: Tue, 13 Mar 2012 08:57:55 +0000*
*Subject: Ahven 2.2 released*
*Newsgroups: rss.cx.ada.planet*

Tero Koskinen recently released version 2.2 of his Ahven unit testing library: http://sourceforge.net/projects/ahven/

"This (2.2) is long overdue bug fix release for Ahven. Ahven.XML_Runner had same bug as Ahven.Text_Runner and this release fixes it. Now your XML test result files should have skipped tests correctly reported. Another bigger change is API documentation generator change from Adabrowse to Sphinx"

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

Download: http://sourceforge.net/projects/ahven/files/latest/download

## ColdFrame UML to Ada translator

*From: Simon Wright <simon@pushface.org>*
*Date: Tue, 27 Mar 2012 17:41:12 +0100*
*Subject: ANN: ColdFrame 20120324*
*Newsgroups: comp.lang.ada*

This announces release 20120324 of ColdFrame, which generates Ada code frameworks from UML models in ArgoUML.

Changes from previous releases can be seen at the Files link.

Project: https://sourceforge.net/projects/coldframe/

Web: http://coldframe.sourceforge.net/

Files: https://sourceforge.net/projects/coldframe/files/coldframe/20120324/

# Ada-related Products

## Rational R1000 series 400

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Fri, 01 Jun 2012 10:02:55 +0200*
*Subject: Software for Rational R1000 series 400?*
*Newsgroups: comp.lang.ada*

The Danish Computing History Association (DDHF) has received a Rational R1000 series 400 from Terma.

I was invited to assist in opening the crate and inspect the hardware and documentation yesterday.

Apparently the disks in the machine have been erased before the machine left Terma, so DDHF is looking for a source for the operating system software for the R1000/400. Since the disks in the machine seem to be plain SCSI disks, we hope that a raw disk copy will be enough to get it up and running again.

Please contact me or Poul-Henning Kamp ("phk" & Ada.Characters.Latin_1.Commercial_At & "phk.freebsd.dk"), if you have any information regarding software for the Rational R1000 series 400.

[photos from the initial inspection at http://datamuseum.dk/wiki/Rational/R1000s400 —sparre]

# Ada and GNU/Linux

## Ada environment on Slackware

*From: "Zhu Qun-Ying" <zhu.qunying@gmail.com>*
*Date: Thu, 19 Apr 2012 10:43:20 -0700*
*Subject: My little work on building an Ada evnvironment on Slackware*
*Newsgroups: comp.lang.ada*

I have done some work to build an Ada development environment on Slackware64 with gcc 4.7.0. It is based on Ludovic's work on Debian, with some tweaks.

I hope it might be helpful to some one that like to do the same thing.

http://zhuqy.wordpress.com/2012/04/15/building-an-ada-development-environment-using-fsf-gcc-for-slackware/

*From: Thomas Løcke <thomas@12boo.net>*
*Date: Fri, 20 Apr 2012 08:18:19 +0200*
*Subject: Re: My little work on building an Ada evnvironment on Slackware*
*Newsgroups: comp.lang.ada*

[…]

Personally I just grab the packages from libre.adacore.com and install them. I've never really felt any need for turning them into specific Slackware packages, as they are so dead simple to get going.

I actually did 5 short videos on the process a while ago:

http://youtu.be/9LcDyUG2loE (part 1 - the compiler)

http://youtu.be/IZCLBKn9-7o (part 2 - florist)

http://youtu.be/N27lK6Fg8Ek (part 3 - XML/Ada)

http://youtu.be/tVFRK2nIPRw (part 4 - GNATColl)

http://youtu.be/wzbKDaLb76s (part 5 - AWS)

*From: qunying <zhu.qunying@gmail.com>*
*Date: Fri, 20 Apr 2012 23:06:17 -0700*
*Subject: Re: My little work on building an Ada evnvironment on Slackware*
*Newsgroups: comp.lang.ada*

As you suggested, I setup a git repository to host the build scripts at https://github.com/qunying/ada-4-slackware

## GNAT on Debian ARM

*From: tonyg <tonythegair@gmail.com>*
*Date: Fri, 9 Mar 2012 08:19:26 -0800*
*Subject: gnat on debian arm*
*Newsgroups: comp.lang.ada*

Does anyone know if gnat is fully ported to debian ARM and its status?

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Fri, 09 Mar 2012 19:00:21 +0100*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

Fully ported to armel (this is ARM in little-endian mode).

Not ported yet to armhf (this is ARM with hardware floating-point).

This is true of both gnat-4.4 and gnat-4.6.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Tue, 13 Mar 2012 07:13:23 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

Interesting thread, very informative, starting at http://lists.debian.org/debian-arm/2012/03/msg00002.html.

The Raspbery Pi uses an ARMv6 processor running in little-endian mode (ARM supports both big-endian and little-endian modes) with hardware floating point. This is quite a unique combination; processors with hardware floating point are usually ARMv7; the armhf port of Debian requires ARMv7.

Debian armel works on the Raspberry Pi but does not take advantage of the floating point hardware. Note that if you're going to program in Ada on such a machine, you might find that 256 MiB of RAM is very limited. You probably don't want gnat-gps or emacs as your IDE on such a machine.

*From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>*
*Date: Tue, 13 Mar 2012 17:36:27 +0100*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

I had thought that 256 MiB is plenty of RAM for editing text and running a compiler. Editing without a "GUI" using a capable text editor, including Emacs, should well be possible.

256 MiB is about the amount of RAM that were supposedly necessary to _translate_ a compiler for some O-O language in the early 1990s. But I am sure I was happily running Editors, including Emacs, in a lot less than that. In fact, I didn't know anyone who had access to a computer with such an amount of RAM.

Running OS/2 on a PCs equipped with that "limited" amount of RAM went rather smoothly, or is my memory blurred?

*From: "Randy Brukardt" randy@rrsoftware.com*
*Date: Tue, 13 Mar 2012 13:04:49 -0500*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

I think your memory of memory is fine. :-) The original version of Janus/Ada worked fine on a 48K Z-80 machine (just a subset of Ada, though).

The first validated (Ada 83) version was for original 640K IBM PCs. Even today, the memory footprint of Janus/Ada never exceeds 16Mb.

Claw programs are "huge" by our standards, but they too tend to have a footprint in the 16Mb range. So 256MB would be plenty for program development (depending on the OS size; we might only use 32MB but Windows of course uses a heck of a lot more).

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Tue, 13 Mar 2012 20:44:30 +0100*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

Maybe I'm biased towards large-scale development. I have fond memories of my old IBM ThinkPad T22 with 256 MiB RAM and a Pentium III processor running at 900 MHz. The performance of that machine should be quite similar to that of a modern Raspberry Pi. Compiling gnat 3.15p (400 kSLOC C, 300 kSLOC Ada) on that machine was OK but when I started work on GCC 4.1 (2.6 MSLOC), a three-stage bootstrap with tests took an entire night. And building the other large package, gnat-gps, took 220 minutes on that machine and 17 minutes on its successor which is stil my current machine. Most of the difference was due to the increase in RAM (256 MiB to 2 GiB).

http://bugs.debian.org/cgi-bin/bugreport.cgi?msg=100;bug=393636

[…]

Oh, you're bringing memories back... I started running OS/2 2.0 when it was released on 4 MiB (very slow) and upgraded my machine to 8 (okay-ish) then 12 then 24 MiB, which *was- smooth. Yes, the architecture was quite good and I was also a fan :)

I feel like a dinosaur now :)

*From: Rolf <rolf.ebert_nospam_@gmx.net>*
*Date: Wed, 14 Mar 2012 03:02:14 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

I run Debian armel on a Seagate Dockstar [1] with only 128MB RAM. It was sold in 2010 for around 20€, unfortunately it isn't available anymore. Emacs runs just fine on that machine. And I remember that I once bootstrapped gcc (4.6.x?) on it (all three stages, no test runs, >> 1 day)

[1] http://ahsoftware.de/dockstar/

*From: Álex R. Mosteo <alejandro@mosteo.com>*
*Date: Wed, 28 Mar 2012 12:49:51 +0200*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

I had the impression that recent gcc versions are quite memory hungry (when doing optimization?). Also particularly gnat when compiling generics. I might be wrong though.

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Tue, 17 Apr 2012 21:15:49 +0200*
*Subject: GNAT on Raspberry Pi (Was: gnat on debian arm)*
*Newsgroups: comp.lang.ada*

Basic text I/O and tasking works fine with the Debian GNAT package on Raspberry Pi. (I just tested on the real thing a few minutes ago.)

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Wed, 18 Apr 2012 13:57:18 +0200*
*Subject: Re: GNAT on Raspberry Pi*
*Newsgroups: comp.lang.ada*

[…]

I've compiled and tested various old Ada applications of mine on a Raspberry Pi running Debian/stable. Use of tasking, text I/O, random numbers, floating point calculations and the POSIX Ada API has been tested. So far I haven't found any problems.

*From: Lucretia <laguest9000@googlemail.com>*
*Date: Wed, 25 Apr 2012 08:41:46 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

Debian ARM is actually arm-eabi, The current ARM GNAT's exception handling uses SJLJ which is not part of the EABI standard; the standard uses ZCX in DWARF2 tables, which has not been ported to ARM yet, I've tried and got stuck/confused.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Wed, 25 Apr 2012 08:48:52 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

IIUC, that means that pure Ada programs work perfectly well but that exceptions cannot cross language boundaries, i.e. if your Ada program uses some C++ library that raises exceptions, then you cannot handle the exception in Ada? Big deal.

Otherwise, could you please elaborate on what you mean by "stuck/confused"?

*From: Lucretia <laguest9000@googlemail.com>*
*Subject: Re: gnat on debian arm*
*Date: Wed, 25 Apr 2012 09:03:11 -0700*
*Newsgroups: comp.lang.ada*

[…]

It'll be slower than ZCX as well, that is a big deal.

> Otherwise, could you please elaborate on what you mean by "stuck/confused"?

Ever looked at the exception code? Understand the exception tables? Nope, me neither.

I posted a patch to gcc ml with my progress a while back, it works to a point, but there needs to be code to set up the actual ARM exception also. I just got confused with it all tbh.

*From: roderick.chapman@googlemail.com*
*Date: Wed, 30 May 2012 03:58:35 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

My Raspberry Pi arrived this morning, so a port of the SPARK Examiner might be on the cards next... :-)

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Wed, 30 May 2012 06:53:40 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

Already done IIUC, simply say:

aptitude install spark

on Debian testing or unstable on your Raspberry Pi

*From: Tero Koskinen <tero.koskinen@iki.fi>*
*Date: Thu, 31 May 2012 18:58:01 +0300*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

[…]

Spark does not seem to work on my Gumstix Overo with Debian sid/armhf:

```
[15:54 gumstix:21] ~/tmp/s
% file /usr/bin/spark
[...]

[15:54 gumstix:22] ~/tmp/s
% spark
zsh: killed     spark

[15:55 gumstix:23] ~/tmp/s
% gdb spark
[...]

Temporary breakpoint 1 (main) pending.
Starting program: /usr/bin/spark
During startup program terminated with
signal SIGKILL, Killed.
(gdb)

% apt-cache show spark
[…]

'checker' works:

[15:56 gumstix:25] ~/tmp/s
% checker
SPARK Proof Checker GPL 2011
Copyright (C) 2011 Altran Praxis Limited,
Bath, U.K.

Please type filename, without extension, in
lowercase, within single
quotes if it is not in this directory, followed by
a full-stop.
FILENAME.vcg and FILENAME.fdl will be
read.
Filename? abc.
No .vcg file of this name exists.
List of .vcg files in current region:
    <THERE ARE NONE>Please try again.
Please type filename, without extension, in
lowercase, within single
quotes if it is not in this directory, followed by
a full-stop.
FILENAME.vcg and FILENAME.fdl will be
read.
Filename? .
errorStream user_input:10:18 Syntax error:
Unexpected end of clause
```

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Fri, 1 Jun 2012 01:02:46 -0700*
*Subject: Re: gnat on debian arm*
*Newsgroups: comp.lang.ada*

> Spark does not seem to work on my Gumstix Overo with

> Debian sid/armhf:

[...]

> % gdb spark

[...]

> (gdb) run

> Starting program: /usr/bin/spark

> During startup program terminated with signal

> SIGKILL, Killed.

Does it work on armel (as opposed to armhf)?

I reported this problem to the spark maintainer, please followup at http://bugs.debian.org/675385.

*From: Tero Koskinen <tero.koskinen@iki.fi>*
*Date: Tue, 13 Mar 2012 20:43:28 +0200*
*Subject: GNAT on armhf*
*Mailing-list: debian-ada@lists.debian.org*

[…]

What is missing from GNAT for armhf platform? I have ARM hardfp ABI (armhf in Debian terms) capable device (Gumstix/OMAP3), and I could do some testing/development if needed.

I don't remember any calling convention specific things in GNAT or its runtime, so is it just a simple matter of compiling the compiler for armhf?

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Tue, 13 Mar 2012 20:54:14 +0100*
*Subject: Re: GNAT on armhf*
*Mailing-list: debian-ada@lists.debian.org*

[…]

There is a specific calling convention, in particular as regards floating point registers, but I think the compiler deals with that by itself. Also, IIUC, the ABI for ARMv7 supports only the Thumb instruction format whereas ARMv6 allows both Thumb and the older, less densely packed, format. It is this calling convention and format that prevent software compiled for armhf from running on ARMv6 hardware. So, I think what's needed at this point is "just" someone to bootstrap the compiler on the appropriate hardware. If you could do that you'd deserve a star on the walk of fame :) I haven't really researched this issue, so take everything I said with a kilogram of salt, as I may be entirely wrong.

After this bootstrapping, it would be nice to tune Ada.Numerics to use BLAS and LAPACK compiled with hardware floating point.

Thanks for your offer. I appreciate it because we're not short of people with ideas, we're short of experts with free time and energy :)

*From: Tero Koskinen <tero.koskinen@iki.fi>*
*Date: Wed, 18 Apr 2012 22:25:34 +0300*
*Subject: Re: GNAT on armhf*
*Mailing-list: debian-ada@lists.debian.org*

[…]

I managed to create some preliminary GNAT binary, which runs on Debian armhf and generates Debian armhf binaries from Ada source code.

I also did a armhf cross-compiler, which runs on 64-bit Fedora x86_64 platform.

Links to gnat binaries:

Cross-compiler, runs on Fedora x86_64: http://iki.fi/tero.koskinen/debian/armhf/ada-armhf-crosstools-on-fedora16_x86_64_host.tar.gz

Actual arm-linux-gnueabihf compiler, runs on Debian armhf: http://iki.fi/tero.koskinen/debian/armhf/ada-armhf-gcc-debian-sid-armhf.tar.gz

This (second link) is compiled on x86_64 using the compiler in the first link.

I haven't managed to do native build (compile armhf GNAT on Debian armhf using above armhf compiler), since gcc compilation dies to segfault on qemu-arm, and I haven't had time to try it on real hardware yet.

In related news, GNAT on Debian armel seems to work pretty ok on Gumstix.

Only problem is that, I cannot compile my modified ACATS test suite (where all tests go into a single binary), since GNAT dies to out-of-memory error after a while (Gumstix has 256MB ram + 100MB swap on sd-card, qemu-arm kills GNAT when it has comsumed more than 400MB memory).

[…]

## Debian transition to GNAT-4.6

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Fri, 13 Apr 2012 08:44:34 +0000*
*Subject: Re: Transition to gnat-4.6 current status*
*Mailing-list: debian-ada@lists.debian.org*

This is a status update about the transition to gnat-4.6. The previous status update is at http://lists.debian.org/debian-ada/2012/02/msg00021.html

| | |
|---|---|
| adabrowse | OK |
| adacgi | OK |
| adacontrol | OK |
| adasockets | OK |
| ahven | OK |
| apq | OK |
| apq-postgresql | OK |
| asis | OK |
| dh-ada-library | OK |
| gnade | OK |
| gnat | OK |
| gnat-gps | OK |
| gprbuild | OK |
| libaws | OK |
| libalog | OK |
| libaunit | OK |
| libflorist | OK |
| libgmpada | OK |
| libgtkada | OK |
| liblog4ada | OK |
| libncursesada | OK |
| libtemplates-parser | OK |
| libtexttools | OK |
| libxmlada | OK |
| libxmlezout | OK |
| music123 | OK |
| narval | |
| opentoken | OK |
| pcscada | OK |
| polyorb | |
| python-gnatpython | OK |
| spark | OK |
| topal | OK |

As you can see, all packages except for polyorb and narval have now completed the transition to gnat-4.6. dh-ada-library is a new package that Nicolas Boulenguez wrote and submitted. No other package build-depends on dh-ada-library yet but this will happen in due course; Nicolas has already made changes to some packages in the version control system so that their next uploads will use dh-ada-library.

So it is time to think about the migration of all packages to testing. polyorb and narval have been removed from unstable and testing[1] per request of the QA people, so they are not blocking the migration of any other packages. It seems the only package blocking the migration is libgtkada2 (ancestor of libgtkada). I will request its removal and hope that the migration to testing takes place soon.

[1] http://bugs.debian.org/662165

Xavier, please tell us whether you plan to revive polyorb and narval in Debian or not.

*From: Svante Signell <svante.signell@telia.com>*
*Date: Fri, 13 Apr 2012 11:16:43 +0200*
*Subject: Re: Transition to gnat-4.6 current status*
*Mailing-list: debian-ada@lists.debian.org*

[…]

What about ghdl? It currently depends on gnat-4.4. I assume it does not build with gnat-4.6. It was built on most buildds more than a year ago.

*From: Xavier Grave <xavier.grave@ipno.in2p3.fr>*
*Date: Fri, 13 Apr 2012 11:50:34 +0200*
*Subject: Re: Transition to gnat-4.6 current status*
*Mailing-list: debian-ada@lists.debian.org*

[…]

I have version compatible with gnat-4.6 for both, but I'm hitting a testsuite problem in polyorb due to the change from Ada to python.

I'm quite busy with some others stuffs for the moment but the packages build fine and as soon as this testsuite problem is solved I think we can upload polyorb and narval to unstable.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Fri, 13 Apr 2012 13:55:35 +0000*
*Subject: Re: Transition to gnat-4.6 current status*
*Mailing-list: debian-ada@lists.debian.org*

[…]

ghdl does not build-depend on gnat, only on gnat-4.4, so it will not block the transition. You may see the lack of a build-dependency on gnat as a policy violation but ghdl is not a library, does not produce a -dev package, and so cannot adversely affect other packages.

Also, ghdl has 10 outstanding bugs, one of them serious (FTBFS with recent gcc while bootstrapping its own gcc back-end), so I'll let the maintainer deal with the package and not bother him further. If the maintainer is MIA, the QA people will remoe ghdl from Debian.

## Ada on Arch Linux

*IRC-network: irc.freenode.net*
*IRC-channel: #Ada*

2012-03-09:19:10 #ada: < oenone> upgrade was smooth: https://aur.archlinux.org/packages.php?ID=54949

2012-03-12:21:58 #ada: < oenone> btw, just finished ncurses-ada pkgbuild:

https://aur.archlinux.org/packages.php?ID=57546

2012-04-05:15:32 #ada: * oni-work uses Arch Linux

2012-04-05:15:35 #ada: < aod> HalfMadDad: i use arch linux ;)

2012-04-11:21:15 #ada: < pamphoon> I use archlinux or openbsd mostly

2012-05-15:16:13 #ada: < hellsend> oni-work> i never got GPS to work on Archlinux :(

2012-05-15:16:13 #ada: < oni-work> oh, why?

2012-05-15:16:13 #ada: < hellsend> the AUR package seems to be broken

2012-05-15:16:13 #ada: < oni-work> https://aur.archlinux.org/packages.php?ID=28415

2012-05-15:16:13 #ada: < oni-work> ?

2012-05-15:16:29 #ada: < hellsend> oni-work> yeah, the compiling of gnat-gps in Archlinux still outputs an error - but it's changed since I'd tried it :D - today it shows : "toolchains-parsers.ads:197:10: current instance must be a limited type - Error 4"

2012-05-26:15:04 #ada: < oenone> fixed gnat-gps package

2012-05-26:15:04 #ada: < oenone> for arch linux

2012-05-26:15:05 #ada: < oenone> seems like a few things changed with update from 4.6.3 to 4.7.0

# Ada and Microsoft

## Building gnat-gpl-2011-avr-windows

*From: "Rego, P." <pvrego@gmail.com>*
*Date: Sun, 8 Apr 2012 20:18:28 -0700*
*Subject: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

I am trying to build the gnat-gpl-2011-avr-windows toolchain in GPS with a simple project (which I do not expect to run on an AVR chip for now). So I have installed on Windows 7 machine the gnat-gpl-2011-i686-pc-mingw32-bin.exe and the AVR gnat-gpl-2011-avr-windows-bin on c:\GNAT\2011.

I configured a GPS project file as

```
project Test is
  package Compiler is
    for Default_Switches ("ada")
        use ("-gnat05");
  end Compiler;

  package Ide is
    for Gnat use "avr-gnat";
    for Gnatlist use "avr-gnatls";
    for Debugger_Command
        use "avr-gdb";
  end Ide;

  for Source_Dirs use (".", "src");
  for Main use ("main.adb");
end Test;
```

When I try to build, it returns me

```
avr-gnatmake -f -d -PC:\test\test.gpr
main.adb
avr-gcc -c -gnat05 -I- -gnatA
C:\test\src\main.adb
avr-gnatbind -I- -x C:\test\main.ali
avr-gnatlink C:\test\main.ali -o C:\test\main
c:/gnat/2011/bin/../lib/gcc/avr/4.5.3/../../../../av
r/bin/ld.exe: cannot find -lc
collect2: ld returned 1 exit status
avr-gnatlink: error when calling
C:\GNAT\2011\bin\avr-gcc.exe
avr-gnatmake: **- link failed.

[2012-04-09 00:13:05] process exited with
status 4 (elapsed time: 00.25s)
```

So I included in GPS project file some options:

```
  package Linker is
    for Default_Switches ("ada")
      use ("-O", "-mmcu=3Davr6",
           "-nostdlib", "-lgcc",
           "-Wl,-mavr6,
            -Tdata=3D0x00800200");
  end Linker;
```

and now when I try to build it returns me

```
avr-gnatmake -f -d -PC:\test\test.gpr
main.adb
avr-gcc -c -gnat05 -I- -gnatA
C:\test\src\main.adb
avr-gnatbind -I- -x C:\test\main.ali
avr-gnatlink C:\test\main.ali -O -
mmcu=3Davr6 -nostdlib -lgcc -Wl,-mavr6,-
Tdata=3D0x00800200 -o C:\test\main
c:/gnat/2011/bin/../lib/gcc/avr/4.5.3/../../../../av
r/bin/ld.exe: avr architecture of input file
`b~main.o' is incompatible with avr:6 output
c:/gnat/2011/bin/../lib/gcc/avr/4.5.3/../../../../av
r/bin/ld.exe: avr architecture of input file
`C:\test\main.o' is incompatible with avr:6
output
collect2: ld returned 1 exit status
avr-gnatlink: error when calling
C:\GNAT\2011\bin\avr-gcc.exe
avr-gnatmake: **- link failed.

[2012-04-09 00:15:00] process exited with
status 4 (elapsed time: 00.25s)
```

Thus, what did I miss in setup? I just want to generate a .hex file for a dummy main.adb.

*From: Simon Wright <simon@pushface.org>*
*Date: Mon, 09 Apr 2012 08:23:07 +0100*
*Subject: Re: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

That looks like an incompatibility between -mmcu=avr6 which you've set in package Linker and the default options used elsewhere.

I _think_ that there may be a way of saying this sort of thing globally, but as is frequently the case the GPR documentation isn't easy to get to grips with (I guess that people don't post bug reports about it enough to let AdaCore know how difficult it can be to use? I know I didn't when I worked for a paying customer).

Is there a worked example with the toolchain?

*From: "Rego, P." <pvrego@gmail.com>*
*Date: Wed, 11 Apr 2012 15:06:38 -0700*
*Subject: Re: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

[…]

No. I could not find any example that I could run yet. But googling today I found this tutorial by Maciej Kucia: http://student.agh.edu.pl/~mkucia/wiki/doku.php?id=avrada

which looks to be very interesting. The tutorial requires GNAT AVR + GPS + WinAVR GNU gcc tools and libraries for AVR + Atmel AVR Studio 5. I use all of them frequently, so that's no problem, but I'd expect that I could use just GNAT AVR + GPS to build the .hex. However sure it can be a very good starting point. I will try it.

*From: Rolf <rolf.ebert_nospam_@gmx.net>*
*Date: Sun, 15 Apr 2012 09:02:31 -0700*
*Subject: Re: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

[…]

AdaCore's AVR compiler comes with a sample application somewhere (I don't remember where).

When using gcc for AVR you have to specify the target MCU at the compile and link steps with the option -mmcu=<name of mcu>. In your example you missed that option in the compile step.

[…]

AVR-Ada /= AdaCore's AVR compiler.

AVR-Ada V1.1 still uses gcc-4.3.x, AdaCore's AVR compiler is based on gcc-4.5. That version has a serious problem if you use locally renamed variables of the MCU's registers. Either use AVR-Ada V1.1 or wait a few days/weeks for AVR-Ada V1.2 which will use gcc-4.7

*From: Brian Drummond <brian@shapes.demon.co.uk>*
*Date: Tue, 17 Apr 2012 16:58:51 +0000*

*Subject: Re: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

[…]

I have just successfully (as far as I know!) built gcc-4.7 prerelease version (20120302) for AVR, and verified it makes the "Blinky" project http://sourceforge.net/apps/mediawiki/avr-ada/index.php?title=USB_Boarduino#Blinky_in_Ada which works with appropriate changes on three platforms.

I have also added a switch and modified the example along the lines :

```
   Sw : Boolean renames
        MCU.PinB_Bits(1);
...
   MCU.DDRB_Bits :=
       (1 => DD_Input,
        others => DD_Output);
...
      LED <= not Sw;
```

and the fact that it works as expected suggests to me that the GCC4.5 bug with renamed registers has been fixed.

One of the platforms I tried, http://www.pjrc.com/teensy/ uses the Atmega32U4 which isn't in the list of AVRAda 1.1 supported devices. (however, mcu=atmega328p worked for this example) Are there any guidelines how to add support for a new CPU, perhaps by modifying the 32u6 sources?

*From: Simon Wright <simon@pushface.org>*
*Date: Tue, 17 Apr 2012 20:29:05 +0100*
*Subject: Re: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

[…]

Is that http://gcc.gnu.org/bugzilla/show_bug.cgi?id=46192 ? If so, it's certainly fixed in 4.7.0 and should be in 4.5, 4.6 too (releases after 21 January 2012; SVN revisions 18336[567]).

*From: Brian Drummond <brian@shapes.demon.co.uk>*
*Date: Tue, 17 Apr 2012 21:19:53 +0000*
*Subject: Re: Building gnat-gpl-2011-avr-windows from scratch in GPS*
*Newsgroups: comp.lang.ada*

That's the one. Glad it's fixed!

# Ada Inside

## SparForte

*From: Master of Magic <koburtch@gmail.com>*
*Date: Thu, 15 Mar 2012 18:32:07 -0700*
*Subject: ANN: SparForte 1.3*
*Newsgroups: comp.lang.ada*

SparForte 1.3

Type : Programming Language

Platforms: Linux i386/x86_64/Alpha (and FreeBSD)

License: GPL

Home URL: http://www.pegasoft.ca/sparforte.html

Downloads: http://www.pegasoft.ca/sparforte-down.html

SparForte is an Ada-based command shell, template engine and scripting language. It natively interprets Bourne shell commands and basic database commands at the command prompt and has an integrated debugger. There are 23 built-in packages including MySQL, PostgreSQL, CGI and Memcache and over 80 example scripts.

[this is the former Business Shell/BUSH —sparre]

## Session Chair Timer

*From: "J-P. Rosen" <rosen@adalog.fr>*
*Date: Mon, 26 Mar 2012 14:50:19 +0200*
*Subject: [Ann] SC_Timer 2.0, the Session Chair Timer*
*Newsgroups: comp.lang.ada*

Adalog is pleased to announce the release of SC_Timer 2.0, with a completely new interface and much more possibilities than 1.0.

In addition, it is a nice example of what can be accomplished with GTK-Ada and Glade (to the non-believers: yes, it was fully designed with Glade ;-) ).

SC_Timer is a very convenient timer for session chairs in conferences who need to manage the presentation time of speakers. The chair enjoys a rich pannel to manage talk time (screenshot) and a secondary display to let the speaker know the remaining time, prompt messages, etc (screenshot). The display turns yellow - for example- ten minutes before time is over, then turns red five minutes before the end. When the presentation time is over, it displays a big red flashing "Off". Alternatively, the timer can be set to count up (from 0 to programmed time) instead of down. Other features allow to manage various special cases (pauses, increasing or reducing talk time...). There is also a regular clock in the upper right corner that tells you the current time.

Of course, the program is really a general count-down timer, and you can use it for all your count-down needs!

[download from http://www.adalog.fr/progs2.htm —sparre]

## New AWS-based website

*From: Maciej Sobczak <see.my.homepage@gmail.com>*
*Date: Tue, 10 Apr 2012 15:10:51 -0700*
*Subject: New AWS-based website*
*Newsgroups: comp.lang.ada*

I'm pleased to announce the launch of a new Ada-based website:

http://www.vetoteka.pl/

This website is intended for veterinary doctors and their patients and is supposed to provide new communication channels between them. The majority of its functionality is hidden behind logins and passwords, but some part of it is available to general public as well - well, at least to those who understand Polish. ;-)

The website was implemented with the following software stack: Debian + PostgreSQL + AWS. This stack has proved to be a very solid foundation and the implementation of the complete website was a very easy-going experience, except for a few small bumps at the beginning of the road. I believe that the support that these technologies give to the programmer have no equivalent in the web development ecosystem. In particular, the myth that dynamically typed languages are better for web due to the fast turn-around (no compilation needed) is irrelevant when the actual turn-around of the Ada-based solution is observed in practice. I can recommend this solution to anybody who attempts to implement a web service from the ground up.

Please feel free to use this website as a positive example of the Ada-based (AWS in particular) web development.

I will be happy to share the insight and experience that was gained during the development of this website - feel free to contact me here or privately in case of any questions.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Wed, 11 Apr 2012 00:42:20 +0200*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

Thanks a lot for that report. I have two questions:

- what interface did you use between Ada and PostgreSQL?

- did you use the Debian packages, GNAT GPL or a supported GNAT?

*From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>*
*Date: Tue, 10 Apr 2012 18:15:35 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

blazedialer.com is also AWS. Not much to look at without a login, I'm afraid.

*From: tonyg <tonythegair@gmail.com>*
*Subject: Re: New AWS-based website*
*Date: Wed, 11 Apr 2012 00:15:56 -0700*
*Newsgroups: comp.lang.ada*

[…]

I am doing a website using the debian gnat, aws, gnade, with Dmitry Kazonovs database access package (which really simplifies the db stuff).

*From: Maciej Sobczak <see.my.homepage@gmail.com>*
*Date: Wed, 11 Apr 2012 00:21:36 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

> - what interface did you use between Ada and PostgreSQL?

My own. The PostgreSQL API is very simple and easy to bind - I had a working thick wrapper from previous experiments and it worked very well.

The other alternative that I would take into account is SOCI-Ada, but to be frank it would be unnecessarily big for what was needed. I strongly oppose any idea of using ODBC-based solutions.

> - did you use the Debian packages, GNAT GPL or a supported GNAT?

All relevant components above the base system (GNAT, PostgreSQL and AWS) are regular Debian packages.

This was actually related to the "initial bump" that I have mentioned - the development started on a different system with most recent AWS version, while the package available on the target machine was a bit older and missing one or two features related to cookie setting or such. Fortunately these are simple functions, and were easily implemented separately.

No other issues were found and the development work was in general problem-free.

*From: Maciej Sobczak <see.my.homepage@gmail.com>*
*Date: Thu, 12 Apr 2012 00:35:46 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[Why choose PostgreSQL? —sparre]

The reasons are its maturity, run-time stability, excellent and readable documentation, stress on data integrity and independence from short-term licensing/politics/ownership issuess (I find other databases to be missing in at least one and sometimes even all of these categories).

Oh, wait - aren't they the reasons to use Ada, too?

I just find that they fit each other very well.

*From: Thomas Locke <thomas@12boo.net>*
*Date: Thu, 12 Apr 2012 09:45:34 +0200*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

Very nice! It's always a pleasure seeing this in the response headers:

Server: AWS (Ada Web Server) v2.7.0w

[…]

Is there any chance of getting to see the source code?

Did you use templates_parser for the HTML, or something else?

*From: Maciej Sobczak <see.my.homepage@gmail.com>*
*Date: Thu, 12 Apr 2012 07:07:43 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

> Did you use templates_parser for the HTML,

No, I don't see any benefit from it (ironically, this is how PHP was born!).

> or something else?

The HTML that is returned to the client is a combination of:

- static constants,

- generated (computed or obtained from the database or various local data structures) content,

- files.

It depends on the expected frequency of changes in the given content, but since each page is composed of elements from different categories, it is always some combination of these. There are no strictly static pages.

I also did not use the AWS session management, as I find it not very convincing. Any nontrivial session-based operation requires the implementation of additional data structures, which can easily accommodate the complete session handling. There is no reason to use multiple layers of mapping data structures for something that is a consistent functional block. Also, no SOAP.

In other words, considering that there are significant functionality areas of AWS which I did not use, I would very welcome a streamlined or core version of the library that provides only the most essential features. Let's call it "AWS Express Edition" or something like that... ;-)

*From: Thomas Locke <thomas@12boo.net>*
*Date: Thu, 12 Apr 2012 16:28:16 +0200*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[AWS templates parser —sparre]

I think it's nice when you have a few dedicated HTML people, who you'd rather not have mucking around in the Ada code. You can agree on a set of tags and leave them to work their magic on the GUI side of things.

At least that's how I've used templating systems in the past (primarily XSLT based). It's a model I feel works very well.

I've yet to use templates_parser for anything "serious". The only thing I've done with it is the AWS status page and some basic tests.

[…]

Ahh yes, the AWS session management system. I too feel that it would be nice if it were a bit more "flexible", for lack of a better word.

[…]

*From: mockturtle <framefritti@gmail.com>*
*Date: Thu, 12 Apr 2012 13:31:15 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

Well, I never thought about it, but now that you say that... Yes, in my (small) experience of SW for web, what made the PHP code an horrible mess was to try to maintain a status between two different visits of the same site.

*From: Maciej Sobczak <see.my.homepage@gmail.com>*
*Date: Fri, 13 Apr 2012 00:35:49 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

I fully agree with it - if there was such a need, I would divide the work between XML served directly from the server (very likely as a combination of sources as described previously) and the XSLT transformations, which would be under responsibility of other dedicated developers. CSS is on the same side.

That's why I see no place for template parsers built into AWS - in other words, between generating HTML directly and generating XML (that is transformed by XSLT on the client side) there is no space left and therefore server-side template parser is a solution to the problem that does not exist.

> I've yet to use templates_parser for anything "serious".

Exactly.

> Ahh yes, the AWS session management system. I too feel that it would be nice if it were a bit more "flexible", for lack of a better word.

I think it does not need to be. Doing it properly (that is, flexibly enough to cover all needs) would be a huge amount of work and it would always be a wrong solution for some purpose. Considering that doing it right in the actual context of the target system is not difficult at all (support for cookies is all that is neededd), there is again no problem to solve with the library-based solution.

In other words - if the session management can be done entirely in memory, then a screenful of code does the job and the library has little added value; and if it requires support from the database, then the general-purpose library feature will not be adequate anyway.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Fri, 13 Apr 2012 01:05:11 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

I too like PostgreSQL but my first choice of database would be SQLite because it avoids all the network and administrative overhead associated with a database server. I would use PostgreSQL as a second choice if I really needed to run the database and the web server on different machines or if there were more than one client connecting to the same database. Is that your case?

(Oh and in case you ask: you do not need ODBC to talk to an SQLite database).

*From: Maciej Sobczak <see.my.homepage@gmail.com>*
*Date: Fri, 13 Apr 2012 06:50:02 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

There are already several clients connecting, as there are some batch processing jobs running in the background. They are managed separately from the main server process, so having them as tasks/threads within the server was not an option.

A "real" RDBMS has lots of useful features that we intend to benefit from and SQLite is not a valid choice.

To be frank, SQLite belongs to the same category of solutions as template parsers mentioned previously - they are not needed in hello world programs and they don't offer anything adequate in complex systems either.

[…]

*From: Thomas_Locke <thomas@12boo.net>*
*Date: Fri, 13 Apr 2012 16:11:29 +0200*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

> That's why I see no place for template parsers built into AWS - in other words, between generating HTML directly and generating XML (that is transformed by XSLT on the client side) there is no space left and therefore server-side template parser is a solution to the problem that does not exist.

Client side XSLT transformations is not a trivial matter, especially not if you're targetting browsers. It's more or less impossible to write XSL that will work the same across a great many browsers (and versions), and you cannot be sure that XSLT is even properly supported by all browsers.

So more often than not, you will have to do the transformation on the server, in which case templates_parser might do the job just as well, while also being a lot faster. Most XSLT processors are heavy beasts.

From my meager tests templates_parser is very fast, so there might just be a few situations where it could come in handy.

> I think it does not need to be. Doing it properly (that is, flexibly enough to cover all needs) would be a huge amount of work and it would always be a wrong solution for some purpose. Considering that doing it right in the actual context of the target system is not difficult at all (support for cookies is all that is neededd), there is again no problem to solve with the library-based solution.

In other words - if the session management can be done entirely in memory, then a screenful of code does the job and the library has little added value; and if it requires support from the database, then the general-purpose library feature will not be adequate anyway.

You're probably right. I guess I'm just too used to thinking about sessions and cookies in the context of PHP. Old (and bad) habits are hard to kill.

Maybe the AWS.Session package could be made a bit more useful if we could register our own create/destroy/read/write procedures?

*From: Shark8 <onewingedshark@gmail.com>*
*Date: Thu, 12 Apr 2012 13:00:23 -0700*
*Subject: Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

Slightly off-topic, but it's my general impression that a lot of cruft/difficulty in web-development stems from the attempt to impose state on what was developed to be stateless. Would you agree?

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Fri, 13 Apr 2012 10:14:43 +0200*
*Subject: [OT] Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

Definitely!

And now that you've got me started; why do most web developers insist on keeping the details of my state hidden inside their system, instead of storing the state in cookies? (It would avoid the whole mess of sessions timing out for no apparent reason. And it would allow the more adventurous visitors to adjust the state as needed.)

*From: Pascal Obry <pascal@obry.net>*
*Date: Fri, 13 Apr 2012 18:09:29 +0200*
*Subject: Re: [OT] Re: New AWS-based website*
*Newsgroups: comp.lang.ada*

[…]

Because cookies are limited in size and are sent with every request. So to save bandwidth it is far far better to not use cookie for session management. It is better to use them as id management and then store the session data into an internal structure in the server.

BTW, using the Web Block support in AWS it is quite easy to build complex templeted system with Ajax with an elegant session named context. Give it a try!

## Saab Electronic Defence Systems Adopts CodePeer

*From: AdaCore Press Center*
*Date: June 12th, 2012*
*Subject: Saab Electronic Defence Systems Adopts CodePeer*
*URL: http://www.adacore.com/press/saab-electronic-defence-systems-adopts-codepeer/*

PARIS, NEW YORK, and GÖTEBORG. June 12th, 2012 – Ada Europe Conference – AdaCore today announced that Saab Electronic Defence Systems (Sweden) has adopted the CodePeer static analyzer tool for use on the GIRAFFE project. This advanced static analysis tool helps developers detect potential run-time and logic errors in Ada programs. By mathematically analyzing every line of software, and considering every possible input and every path through the program, CodePeer can be used very early in the development lifecycle to identify problems when defects are much less costly to correct.

CodePeer is fully integrated into the GNAT Pro development environment and comes with a number of complementary static analysis tools common to the technology – a coding standard verification tool (GNATcheck), a source code metric generator (GNATmetric), a semantic analyzer, and a document generator that can be invoked through the GNAT Programming Studio (GPS) Integrated Development Environment (IDE).

[…]

# Ada in Context

## Ada 2012: for … of … loop

*From: Martin <martin@thedowies.com>*
*Date: Tue, 29 May 2012 07:29:41 -0700*
*Subject: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

This addition to Ada is brilliant...I've just been refactoring some Ada 2005 into Ada 2012 and this just makes the code so much clearer! No more nested subprograms (of nested subprograms (of nested ...)).

*From: Martin <martin@thedowies.com>*
*Date: Wed, 30 May 2012 01:10:22 -0700*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

[…]

Here's a very small example... what we have now:

```
procedure Save (
   This : in out Configuration;
   Filename : String;
   Header_Comments:
        String_Vectors.Vector :=
          String_Vectors.Empty_Vector;
   Include_Last_Saved_Time :
        Boolean := False) is
  use type String_Vectors.Vector;
  File : File_Type;
begin
  begin
    Create (File, Out_File, Filename);
  exception
    when others =>
      Open (File, Out_File, Filename);
  end;
  for Comment of Header_Comments
      loop
    Put_Line (File,
              Comment_Str & Comment);
  end loop;
  if Include_Last_Saved_Time then
    Put_Line (File, Comment_Str &
                    "Last saved: " &
                     Image (Date => Clock));
  end if;
  for Section of This.Sections loop
    Put_Line (File, "[" &
      To_String (Section.Name) & "]");
    for Pair of Section.Pairs loop
      Put_Line (File,
                To_String (Pair.Key) &
                   "=" &
                   To_String (Pair.Value));
    end loop;
  end loop;
  Close (File);
exception
  - - implementation detail
end Save;
```

what we used to have (mocked up!):

```
procedure Save (
          This : in out Configuration;
          Filename  : String;
          Header_Comments :
             String_Vectors.Vector :=
               String_Vectors.Empty_Vector;
          Include_Last_Saved_Time :
             Boolean := False) is
  use type String_Vectors.Vector;
  File : File_Type;
  -----------
  -- Write --
  -----------
  procedure Write (
     C : Section_Vectors.Cursor) is
    -------------------
    -- Write_Comment --
    -------------------
    procedure Write_Comment (
         C : String_Vectors.Cursor) is
    begin
      Put_Line (File, Comment_Str &
           String_Vectors.Element (C));
    end Write_Comment;
    -----------
    -- Write --
    -----------
    procedure Write (
      C : Key_Value_Pair_Vectors.Cursor)
    is
      Pair : constant Key_Value_Pair :=
      Key_Value_Pair_Vectors.Element (C);
    begin
      Put_Line (File, To_String (Pair.Key) &
              "=" & To_String (Pair.Value));
    end Write;

    This_Section : constant Section :=
         Section_Vectors.Element (C);
  begin
    Put_Line (File, "[" &
            To_String (This_Section.Name)
            & "]");
    This_Section.Pairs.Iterate
                              (Write'Access);
  end Write;

begin
  begin
    Create (File, Out_File, Filename);
  exception
    when others =>
      Open (File, Out_File, Filename);
  end;
  Header_Comments.Iterator
              (Write_Comment'Access);
  if Include_Last_Saved_Time then
    Put_Line (File, Comment_Str &
          "Last saved: " &
          Image (Date => Clock));
  end if;
  This.Sections.Iterate (Write'Access);
  Close (File);
exception
  -- implementation detail
end Save;
```

That's 54 lines down to 29 lines, no 'Access and it's perfectly clear what's happening and it happens in a single subprogram. No need to talk about Cursor or calls to Element (C)...

...It also enhances the value of other nested subprograms - they're probably doing something more interesting, so I don't mind the comment-box headers for them as much as I used to.

All thanks to a little syntatic sugar.

To my eyes, this a big win.

*From: Jeffrey Carter <spam.jrcarter.not@spam.not.acm.org>*
*Date: Wed, 30 May 2012 12:30:21 -0700*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

[…]

While this is a nice feature, I don't think it's worth the added language complexity.

In general, using "for" loops will be clearer than using iterators. In the case of vectors (or unbounded arrays, to use their correct name), one can iterate over them using "for" loops, and the use of iterators for an array abstraction is a questionable practice:

```
for I in Header_Comments.First_Index ..
      Header_Comments.Last_Index loop
   Put_Line (File, Comment_Str &
         Header_Comments.Element (I) );
end loop;

if ... then
   ...
end if;

for I in This.Sections.First_Index ..
      This.Sections.Last_Index loop
   Section := This.Sections.Element (I);
   Put_Line ...

   for J in Section.Pairs.First_Index ..
        Section.Pairs.Last_Index loop
     Pair := Section.Pairs.Element (J);
     Put_Line ...
   end loop;
end loop;
```

I think this is equally clear and requires no additional language complexity.

Had your example used sets or maps it would have been a stronger argument.

*From: Pascal Obry <pascal@obry.net>*
*Date: Wed, 30 May 2012 22:54:14 +0200*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

[…]

No I think you missed one point. In the case of:

```
for Item of <array or containers> loop
   ...
end loop;
```

Item can be read and written in-place. This can be achieved currently without a callback.

*From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>*
*Date: Thu, 31 May 2012 00:26:02 +0200*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

[…]

To my eyes also. It is really nice that Iterate and the "loop body subprograms" are not torn apart. That seems to outweigh the loss of naming opportunities.

I'm still unsure if sweetening adds clarity when one has to write for "the other side" of syntax sugar. For example, when providing a concrete Forward_Iterator from Ada.Iterator_Interfaces. However,

because writing a Forward_Iterator seems to require more types (and packaging) in place of juggling subprograms from the container types, that might be a win too, even at the price of added LOC.

As an example, I have tried a "functional" Reduce function written in 2005 style and in 2012 style, special in that it is required to operate on a span of the values only, not the whole container, mimicking Floor and Ceiling. In pseudo code,

```
"+"/(First .. Last),
```

While the 2012 version of Reduce is pleasantly clear and a little shorter, writing the "backstage stuff" had, at first, seemed to spoil the effort.

In Ada 2012, there is an Iterate function in the 2012 Vectors that takes a parameter Start : Cursor, but not a corresponding Finish : Cursor.

So I made a Forward_Iterator that would know Start and Finish. It has a construction function and overriding First and Next. This made the whole program quite a bit longer. Not necessarily worse, I think, but longer. Perhaps more "rigorous", or "typish"?

I hope the any unnecessary lengthening is caused by my limited understanding. Or that the type centric approach of 2012 Forward_Iterator is better, more clear, and more modular in the end.

Package Iterations_05 below is the 2005 version of the Reduce part. I think the approach follows standard STL style programming.

```
generic
   type Element_Type is private;
   type Cursor is private;
   with function Next (Position : Cursor)
return Cursor is <>;
   with function Element (Position : Cursor)
return Element_Type is <>;
package Iterations_05 is

   generic
     Zero : in Element_Type;
     with function Binop (
            Left, Right : in Element_Type)
          return Element_Type;
   function Reduce_05 (
            First, One_After : Cursor)
          return Element_Type;
end Iterations_05;

package body Iterations_05 is
   function Reduce_05 (
            First, One_After : Cursor)
          return Element_Type is
     Result : Element_Type;
     Position : Cursor;
   begin
     Result := Zero;
     Position := First;
     loop
       exit when Position = One_After;
       Result := Binop (Result,
                    Element (Position));
       Position := Next (Position);
     end loop;
     return Result;
   end Reduce_05;
end Iterations_05;
```

After instantiating the two generics, one picks cursors for the endpoints of the "intervals" to be taken from a container. (At 1 and 50, and 51 and 100 in this case.)

```
with Ada.Text_IO;
with Iterations_05;
with Num_Types, Num_Vecs;
procedure Test_Iterations_05 is

   use Num_Vecs;

   package Functional is
     new Iterations_05
      (Element_Type => Num_Types.Nat,
       Cursor => Num_Vecs.Cursor);

   use type Num_Types.Nat;

   function Sum is
      new Functional.Reduce_05
       (Zero => 0,
        Binop => "+");

   package Num_IO is
      new Ada.Text_IO.Integer_IO
                          (Num_Types.Nat);

   List : Num_Vecs.Vector;
begin
   for K in Num_Types.Nat
       range 1 .. 100 loop
     List.Append (K);
   end loop;
   Num_IO.Put (Sum (First     => First (List),
                One_After => Next (To_Cursor
                                     (List,  50))));
   Num_IO.Put (Sum (First     => To_Cursor
                                     (List, 51),
                    One_After => Next (
                                     Last (List))));
end Test_Iterations_05;
```

That's it.

The 2012 version of the "functional" part Iterations_2012 is just a little shorter than its 2005 counterpart. The body of the Reduce function uses a sweetened for loop:

```
with Ada.Iterator_Interfaces;
generic
   type Element_Type is private;
   with package Iteration is
     new Ada.Iterator_Interfaces (<>);
     with
      function Element (Position :
                        Iteration.Cursor)
       return Element_Type;
package Iterations_2012 is

   generic
     Zero : in Element_Type;
     with function
        Binop (Left, Right : in Element_Type)
        return Element_Type;
   function Reduce_2012 (
     Span : Iteration.Forward_Iterator'class)
     return Element_Type;

end Iterations_2012;

package body Iterations_2012 is
   function Reduce_2012 (
      Span : Iteration.Forward_Iterator'class)
    return Element_Type is
     Result : Element_Type;
   begin
     Result := Zero;
     for k in Span loop
       Result := Binop (Result, Element (k));
     end loop;
     return Result;
   end Reduce_2012;

end Iterations_2012;
```

Since a sweetened for loop should be used, I though that passing a "slice" of a container could only be achieved by a Forward_Iterator. Is this correct? Note the lengthy package called Splitting that provides it.

```
with Ada.Iterator_Interfaces;
with Num_Types, Num_Vecs;
with Iterations_2012;
with Ada.Text_IO;
procedure Test_Iterations_2012 is
   use Num_Vecs;

   package Split_Iterators is
     new Ada.Iterator_Interfaces
        (Cursor => Num_Vecs.Cursor,
         Has_Element =>
           Num_Vecs.Has_Element);

   package Splitting is
     use Split_Iterators;

     type Split_Iterator is limited
       new Split_Iterators.Forward_Iterator
         with record
           First, One_After:Num_Vecs.Cursor;
         end record;

     function Make_Iterator (
       First, One_After : Num_Vecs.Cursor)
       return Split_Iterator;
     overriding function First (
       Object : Split_Iterator) return Cursor;
     overriding function Next (
       Object : Split_Iterator; Position: Cursor)
       return Cursor;
   end Splitting;
```

```
   package body Splitting is
     function Make_Iterator (
       First, One_After : Num_Vecs.Cursor)
       return Split_Iterator is
     begin
       return (Split_Iterators.Forward_Iterator
                with First, One_After);
     end Make_Iterator;

     overriding function First (
       Object : Split_Iterator) return Cursor is
     begin
       return Object.First;
     end First;

     overriding function Next (
        Object : Split_Iterator;
        Position: Cursor) return Cursor is
     begin
       if Position = Object.One_After then
         return Num_Vecs.No_Element;
       else
         return Num_Vecs.Next (Position);
       end if;
     end Next;
   end Splitting;

   package Functional is
     new Iterations_2012
     (Element_Type => Num_Types.Nat,
      Element => Num_Vecs.Element,
      Iteration => Split_Iterators);

   use type Num_Types.Nat;

   function Sum is
     new Functional.Reduce_2012
     (Zero => 0,
      Binop => "+");

   package Num_IO is
     new Ada.Text_IO.Integer_IO (
       Num_Types.Nat);

   List : Num_Vecs.Vector;
begin
   for K in Num_Types.Nat
       range 1 .. 100 loop
     List.Append (K);
   end loop;

   Num_IO.Put (Sum (Splitting.Make_Iterator
     (First     => First (List),
      One_After => Next (
            To_Cursor (List,  50)))));

   Num_IO.Put (Sum ( Splitting.Make_Iterator
     (First     => To_Cursor (List, 51),
      One_After => Next (Last (List)))));
end Test_Iterations_2012;
```

Is Splitting written in the way we should, when using Ada 2012?

*From: Martin <martin@thedowies.com>*
*Date: Thu, 31 May 2012 07:09:38 -0700*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

[…]

Aside from the benefits Pascal mentions, the new syntax offers both readability *and writability* which is a bit of a departure for the language ;-)

*From: "Randy Brukardt" <randy@rrsoftware.com>*
*Date: Wed, 6 Jun 2012 19:19:53 -0500*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

> "In Ada 2012, there is an Iterate function in the 2012 Vectors that takes a parameter Start : Cursor, but not a corresponding Finish : Cursor."

You don't really need one, because you can use "exit" in the for loop in order to stop the iteration early. And doing that explicitly does not suffer from the confusion that could occur if Finish is not actually after Start (in such a case, you'll probably iterate all the way to the end of the container; checking that Finish is somewhere after Start in the container is expensive in general so it isn't something that we wanted to do. So we left this operation out on purpose.

This does require using the cursor version of the iterator, so it is bit more wordy:

```
for Cur in My_Vector.Iterate(Start) loop
   -- Operation code here, probably using
      My_Vector.Reference(Cur)
   --  to access and modify the element
       in place.
   exit when Cur = Finish;
end loop;
```

The advantage here is that it is obvious what happens if Finish doesn't designate an element after Start in My_Vector: the iteration just goes to the end of My_Vector and stops.

> "So I made a Forward_Iterator that would know Start and Finish. It has a construction function and overriding First and Next. This made the whole program quite a bit longer. Not necessarily worse, I think, but longer. Perhaps more "rigorous", or "typish"?"

Certainly possible, but not the recommended way to write such an iteration. Do you check for the error case of Finish not following Start in the container?

*From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>*
*Date: Thu, 07 Jun 2012 14:42:05 +0200*
*Subject: Re: Ada 2012 : In praise of 'for ... of ... loop'...*
*Newsgroups: comp.lang.ada*

In the case of generic algorithms, the situation is a bit reversed, since "it can stop the iteration early", where "it" is the generic algorithm that has the for loop in its body. Using Ada 2005, the generic algorithm need knowledge of a Cursor type and an Element function:

```
      loop
        exit when Cur = One_After;
        Result := Binop (Result, Element
(Cur));
        Cur := Next (Cur);
      end loop;
```

Using Ada 2012, the following looks nice, but might not (currently?) be possible, because Cursor (detailed below) is an incomplete tagged formal type of Iterator_Interfaces, and can't be used here:

```
      for Cur in Span loop     -- Span :
                  Forward_Iterator'Clas
        exit when Cur = One_After;
        Result := Binop (Result,
                    Element (Cur));
      end loop;
```

It also felt a little odd to have to pass both a Forward_Iterator and a Cursor so that the loop could be stopped early. If the following attempt is a possible solution at all.

The full algorithm below is adapted for use with Forward_Iterator as you suggested (such as the one returned by function Vectors.Iterate) IIUC. GNAT complains about invalid use and premature use of Iteration.Cursor, more so in case I write "use type Iteration.Cursor". (I have omitted testing for One_After "<=" Start.)

```
with Ada.Iterator_Interfaces;

generic
   type Element_Type is private;
   with package Iteration is
     new Ada.Iterator_Interfaces (<>);
     with function Element (
       Position : Iteration.Cursor)
     return Element_Type;
package Iterations_2012 is
   generic
     Zero     : in Element_Type;
     with function Binop (
          Left, Right : in Element_Type)
     return Element_Type;
   function Reduce_2012  (
      Span : in
           Iteration.Forward_Iterator'Class;
      One_After : in Iteration.Cursor)  --!
   return Element_Type;
   --  Apply Binop to the sequence of
       elements in Span, excluding every
   --  element starting at One_After
end Iterations_2012;

package body Iterations_2012 is

   function Reduce_2012
     (Span : Iteration.Forward_Iterator'Class;
      One_After       : Iteration.Cursor)
   return Element_Type is
     Result : Element_Type;
```

```
    use Iteration;  --use type Iteration.Cursor;
  begin
    Result := Zero;
    for Cur in Span loop
          exit when Cur = One_After;  --!
      Result := Binop (Result,
                       Element (Cur));
    end loop;
    return Result;
  end Reduce_2012;

end Iterations_2012;
```

A typical scenario I have in mind is divide and conquer: there are tasks that will perform some algorithm on (sub)sequences of elements from just any container.

## Linking in an externally compiled .o file with gnatmake

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Mon, 11 Jun 2012 12:06:17 +0200*
*Subject: Linking in an externally compiled .o file with gnatmake?*
*Newsgroups: comp.lang.ada*

I'm working on a binding to the Linux sound API ALSA (http://repositories.jacob-sparre.dk/alsa-binding). As a part of the binding, I need to access a C macro. I do that by turning the macro into a C function:

```
% cat alsa_macros.c
#include <alsa/asoundlib.h>
void allocate_alsa_hardware_parameters
(snd_pcm_hw_params_t **hwparams_ptr) {
  snd_pcm_hw_params_alloca
(hwparams_ptr);
};
```

Compiling "alsa_macros.c" to "alsa_macros.o" is not a problem, but how can I get gnatmake to link "alsa_macros.o" into the final program? Or maybe into "sound-alsa.o" as it is package Sound.ALSA which imports the function.

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Mon, 11 Jun 2012 12:14:29 +0200*
*Subject: Re: Linking in an externally compiled .o file with gnatmake?*
*Newsgroups: comp.lang.ada*

[…]

Adding "alsa_macros.o" to the linker switches does the job:

```
  package Linker is
    for Default_Switches ("ada")
      use ("-lasound", "alsa_macros.o");
  end Linker;
```

Thanks to tkoskine on the Ada IRC channel for a quick response.

## Ada on embedded devices

*From: Natasha Kerensikova <lithiumcat@gmail.com>*
*Date: Fri, 27 Apr 2012 18:56:10 +0000*
*Subject: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

I have been desiring getting a try at developing for embedded platforms for a while now.

Moreover, I just happen to own a device that probably qualifies as embedded, with two ARM CPUs (one ARM7 and one ARM9), 4 MB RAM and a few DSPs. The device is known as Nintendo DS lite.

So how do I compile code suitable to run on such a device?

I haven't been able to find anything that perform such a task, but I have missed it?

As far as I can tell, devkitARM is a toolchain derived from GCC that targets my hardware (it seems there is also some libraries to deal with DSPs and stuff, but I can care about this later on). On the other end, gnatdroid seems to successfully translate Ada code into binary that can be fed to ARM CPUs. So unless I'm missing something, there is nothing new to discover, I would only need to merge both derivations.

Would anyone have an estimation or a bound on how difficult it can be?

I have never built a compiler myself (except gnat-aux ports, but it just works, so I have no actual experience of making a compiler code build), so I'm not sure exactly how much of a task it is.

Maybe I should start with smaller steps, like writing C stuff going through devkitARM, and then only start aiming at Ada?

*From: Stephen Leake <stephen_leake@stephe-leake.org>*
*Date: Sat, 28 Apr 2012 03:19:40 -0400*
*Subject: Re: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

[…]

It should be straight-forward to include Ada in that toolchain. The hard part will be the Ada runtime library; it assumes some operating system, similar to POSIX. It is possible to use no runtime, but then you lose some features of Ada (exceptions, fixed point, tasking, new/free, etc).

Is there an OS on the Nintendo?

[…]

gnatdroid is intended for Android, a specific operating system. It might be useful as an example.

[…]

If Nintendo has a decent operating system, Ada should Just Work (similar to the ports you describe doing)

But if it doesn't, the level of difficulty really depends on your level of experience; since you seem to be new at configuring compilers, it will be hard and confusing :(.

Definitely set a goal of a runtimeless compiler, and see if you can make that work. Then think about what parts of the Ada runtime you really need.

> Maybe I should start with smaller steps, like writing C stuff going through devkitARM, and then only start aiming at Ada?

That can be useful just to understand how all the tools work together. In particular, writing a simple program in C, and the same in Ada, can be instructive in finding out why the Ada compiler isn't doing the right thing.

But it also leads to the Dark Side :). (as in, "I know I can do this simple program in C; I'll put off porting Ada just a little longer").

*From: Brian Drummond <brian@shapes.demon.co.uk>*
*Date: Sat, 28 Apr 2012 09:22:14 +0000*
*Subject: Re: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

[…]

Me too, though I haven't had all the success I would like, so far! But I hope a beginner's viewpoint (at building gcc) may be of some interest...

In my case I want to be able to build the same version of gcc for native x86/x64, AVR (Arduino etc), Android, Raspberry Pi. I have managed the first two of these last month,(got stuck on Arm/Android last summer, moving house keeps getting in the way)

SO some comments that may help...

apologies if the details are inaccurate, my notes are on another machine.

(1) Building GCC "should be straightforward" as Stephen Leake says. However it probably took me three weeks for x86 and AVR...

(a) Read the prerequisites but they may not be entirely accurate, and differ for different gcc versions, and Ada support in crosscompilers is a less explored corner...

My starting point, gcc_4.7.0_rc1_20120302 failed to build with one version of mpfr, and then failed another "prerequisite" (ppl or ploog) when I turned on Ada language support. I backtracked a version on the first, and omitted the second (which turned out to be an optional prerequisite).

The prerequisites gmp, mpc, mpfr are essential, others seem to be optional so far.

(b) Do not build in the source dir. The FAQ isn't kidding about this.

(c) Script the process. Helps to repeat it in the same order each time. I made many errors not doing so. Safest to delete the "build" directory each time (but slowest).

(d) Build a native version first. Then build it again, using itself as the compiler. If it can build itself, THEN use it to build a crosscompiler. Build the crosscompiler with C support only first, then build the C library, then turn on Ada support and build again.

(e) If it goes wrong, the gcc build system is so obscure that I found it difficult to make any progress, and sometimes had to do things the completely *wrong- way to move forward, intending to learn the right way later. e.g. patching a Makefile each time from the script, because I don't understand how to make the autoconfigure tools generate it correctly in the first place...

(f) Building gcc crosscompilers with Ada support definitely feels lonely...

I must say gcc4.7 rc1 seemed to build with fewer problems than the gcc4.6 version I was trying to build last summer.

But having a working compiler (or using gnatdroid) is only part of the problem...

[ARM7 + ARM9 + 4 MB RAM + a few DSPs = Nintendo DS lite —sparre]

I know nothing about the runtime support on that system. There are probably 2 levels of problem : (1) how to get a basic executable to run (gnatdroid produces console apps that run successfully on the Android) and (2) how to tap into the machine's higher level facilities, gamepad, GUI etc. This still seems to be a sticking point with Gnatdroid/Android let alone the Nintendo...

[…]

If its source is available, building it with Ada support may be possible. But what version of gcc? Bad BAD bad things happen when building gcc4.3.3 with gcc4.5 for example - like negative enumerations (which 4.3 uses and 4.5 rejects as errors!

Or the runtime system (libc, interface to GUI etc) may be portable to a newer GCC. Or you may be able to substitute the right version of GCC, separately built, with Ada support. But any of these may fail to build unless you can find or recreate the correct patches. IMO gcc is frighteningly fragile...

> On the other end, gnatdroid seems to successfully translate Ada code into binary that can be fed to ARM CPUs. So unless I'm missing something, there is nothing new to discover, I would only need to merge both derivations.

A promising approach, I think. Where I fell over was combining gnatdroid's output into the Android NDK build system to use anything more sophisticated than the console - i.e. open a simple window and display "Hello World". (I could get it through the linker, but the resulting executable didn't work)

I can imagine similar problems with the Nintendo.

> Maybe I should start with smaller steps, like writing C stuff going through devkitARM, and then only start aiming at Ada?

Definitely. This is an essential part of the process, so you have a "known good" chain to compare the Ada chain to, step by step. Can I substitute my Ada .o for this C .o into the linker? If not, what is different between them?

> Would anyone have an estimation or a bound on how difficult it can be?

Difficult. But IMO the more people building Ada for embedded platforms, the easier it will become, if we share information and feed back fixes into the process. (Which I have not been doing, mainly because what I see as an obstacle, probably arises from my own blockheadedness rather than a genuine bug in gcc. And I have felt reluctant to submit them to sanity checks before passing any potential bugs upstream - mainly because there don't seem t be many of us trying it, so where to submit them? Clearly there are some, as Jacob Sparre Andersen recently reported success on Raspberry Pi If you think my notes/scripts so far may be useful, I'll pass them on.)

Consider having another platform available as another way of moving forward. (This works on the Raspberry Pi but not the Nintendo... what is different between them?)

*From: Natasha Kerensikova <lithiumcat@gmail.com>*
*Date: Sat, 28 Apr 2012 13:43:18 +0000*
*Subject: Re: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

> It should be straight-forward to include Ada in that toolchain. The hard part will be the Ada runtime library; it assumes some operating system, similar to POSIX. It is possible to use no runtime, but then you lose some features of Ada (exceptions, fixed point, tasking, new/free, etc).

Well, that's encouraging for a start :-)

Is there some "official" list of what does not have to be supported on an RTL-less system? The C Standard has the notion of "freestanding" and "hosted" environments, which basically means without or with libc (which is quite weaker than POSIX); is there something similar in Ada RM? Or is it up to the compiler provider to decide what belong to RTL and what is code generated on bare metal?

For example I'm a bit surprised to see fixed point in the list, when I would have naively thought appropriate inline code generation would be enough (exactly like when I did fixed point "by hand" in 386 assembly).

> Is there an OS on the Nintendo?

I would have to double check, but as far as I remember no. There is only a bootloader that transfers binary code from cartridge storage to memory and jump to it, and that's it.

An even more fuzzy memory of mine says Nintendo sells a SDK that comes with most of what you would expect from an OS, but that's not exactly within hobbyist financial reach.

On the other hand, devkitARM comes with a libnds that might take care of a reasonable amount of stuff.

>> On the other end, gnatdroid seems to successfully translate Ada code into binary that can be fed to ARM CPUs.

> gnatdroid is intended for Android, a specific operating system. It might be useful as an example.

That's all I expected from it, really.

My reasoning was that devkitARM is an example of

C --(1)--> GCC internals --(2)--> Nintendo DS

while gnatdroid is an example of

Ada --(3)--> GCC internals --(4)--> native Android

What I need is just arrows (2) and (3), and I chose gnatdroid for the argument to lower the risk of stumbling on platform-related variation in "GCC internals" part. Both are ARM code, and that's the closed target I have found.

[…]

That was most definitely my plan. First make RTL-less Ada run (which might not be that easy considering "proving" something runs involves an output, that require going through DSPs or direct access to LCD or speakers), then bind divkitARM's libnds, and then the parts of libada I miss, then celebration about the projet reaching such an incredibly mature stage, then the rest of libada.

[…]

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Sun, 29 Apr 2012 09:55:47 -0400*
*Subject: Re: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

> Is there some "official" list of what does not have to be supported on an RTL-less system? […]

No. Some of the annexes are optional, but that's not really analogous to C's "freestanding". An "Ada" without any run-time libraries is almost certainly not Ada -- it might be a subset of Ada.

> ... Or is it up to the compiler provider to decide what belong to RTL and what is code generated on bare metal?

Yes. The same is true of C, by the way, and I suppose any other high-level language. Language definitions are in the business of telling compiler writers what to do, not how to get it done.

Note that the line is kind of fuzzy in the case of GNAT, because some features are implemented as RTL calls, but those calls can sometimes be inlined.

*From: Rugxulo <rugxulo@gmail.com>*
*Date: Sun, 29 Apr 2012 11:29:10 -0700*
*Subject: Re: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

[…]

You could always check this mailing list: http://sourceware.org/ml/crossgcc/

[…]

Okay, so here's the real tip I wanted to pass along, though it's far from secret: FreePascal's website claims that their ARM support covers Linux, WinCE, GBA, and NDS. So there. I know it's not quite "Ada" per se, but close enough to be comfortable (or at least moreso than C, right?). And it doesn't need (barely any) GNU tools to build or use, i.e. no complex GCC / POSIX muck (no offense!). So it should be self-contained and easy to use (though I've not tried, I neither have nor want a NDS).

http://sourceforge.net/projects/freepascal/files/NDS/2.6.0/

arm-nds-fpc-2.6.0.i386-win32.zip
2011-12-31 15.6 MB

Hmmm, I'm surprised it doesn't have Linux host binaries, but I blindly assume it's easy to build as FPC is pretty darn portable.

*From: Micronian Coder <micronian2@gmail.com>*
*Date: Mon, 30 Apr 2012 22:34:37 -0700*
*Subject: Re: Ada on Nintendo DS ?*
*Newsgroups: comp.lang.ada*

Have you considered looking into RTEMS? http://wiki.rtems.org/wiki/index.php/RTEMSAda

There is a Nintendo DS BSP (http://wiki.rtems.org/wiki/index.php/Nds) that has support for some low level functions (e.g. LCD access). The Ada compiler for RTEMS was updated quite often last time I tried it (at least 3 years ago). When I experimented with Ada on RTEMS I built a cross compiler for the Nintendo GameBoy Advance on a Slackware Linux machine. Instead of running on real hardware, I ran the binary output in a GBA emulator (I forgot which one, and it wasn't maintained any longer when I found it). I only got as far as printing a Hello World program and translated a C pong game example into Ada that ran very well. There was an issue with exceptions not getting caught, but it looks like that may have been fixed in more recent version of GCC for RTEMS (see http://gcc.gnu.org/bugzilla/show_bug.cgi?id=3D35143 )

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: Thu, 03 May 2012 14:37:28 +0200*
*Subject: Ada on embedded devices (Was: Ada on Nintendo DS ?)*
*Newsgroups: comp.lang.ada*

[…]

You might want to follow the AVR-Ada mailing list/newsgroup (nntp+news.gmane.org:gmane.comp.hardware.avr.ada), even if your primary embedded platform is a different one.

[…]

- MSP430 (TI Launchpad) - GCC but no Ada reported yet (according to Google).

One known Ada project targeting the MSP430 used AdaMagic as a front-end to a C compiler targeting MSP430. Not my favourite solution, but definitely a possible solution.

- Atmel (Arduino etc.) - AVR-Ada works but without much of a run-time system.

We are actively using AVR-Ada on a commercial project I'm working on.

- ARM (various boards) - GNAT works (apparently) flawlessly on Debian/Squeeze on Raspberry Pi. Tero Koskinen is doing some work to get Gumstix modules to the same level of Ada support as the Raspberry Pi. ACT has a cool demo of GNAT with a Ravenscar run-time running on Mindstorms NXT (I have some LEGO here in the office too ;-).

[…]

*From: Brian Drummond <brian@shapes.demon.co.uk>*
*Date: Fri, 4 May 2012 09:12:41 +0000*
*Subject: Re: Ada on embedded devices (Was: Ada on Nintendo DS ?)*
*Newsgroups: comp.lang.ada*

[working AVR-Ada —sparre]

Good to know. I have the "Blinky" example running on 3 platforms (Arduino, Teensy (where the LED is on a different port) and the Evil Mad Scientist breadboard... Working on stepper control through the Arduino MotorShield.

[…]

R-Pi is getting a LOT of publicity at the moment, and it's a shame that Ada isn't riding on that wave yet. Especially since Ada has a good role as a teaching language, and the R-Pi is marketed as an educational product.

Don't forget Android. A touchscreen tablet could replace a lot of embedded systems in industry (DRO/CNC readouts/controllers for a start) Gnatdroid on DragonflyBSD is a good start but its executables are not really integrated with the Android system.

*From: Tero Koskinen <tero.koskinen@iki.fi>*
*Date: Fri, 4 May 2012 19:36:59 +0300*
*Subject: Re: Ada on embedded devices (Was: Ada on Nintendo DS ?)*
*Newsgroups: comp.lang.ada*

[…]

If you end up ordering something from Adafruit, don't forget to buy Ada Lovelace stickers:

https://www.adafruit.com/products/701

https://www.adafruit.com/products/696

*From: Tero Koskinen tero.koskinen@iki.fi*
*Date: Fri, 4 May 2012 19:42:20 +0300*
*Subject: Re: Ada on embedded devices (Was: Ada on Nintendo DS ?)*
*Newsgroups: comp.lang.ada*

[…]

To be more precise: Gumstix Overo devices can run same Debian armel distribution as Raspberry Pi devices. In addition, Overo can run Debian armhf (which Raspberry Pi cannot).

Debian armhf does not contain GNAT yet. I have been working on the issue, but compiling full GCC package with all languages is somewhat slow on 720MHz machine... (I have GNAT for armhf done, but it is cross-compiled on x86_64 machine, fully native build is not done yet.)

*From: Lucretia <laguest9000@googlemail.com>*
*Date: Sat, 5 May 2012 07:25:34 -0700*
*Subject: Re: Ada on embedded devices (Was: Ada on Nintendo DS ?)*
*Newsgroups: comp.lang.ada*

If you go to https://github.com/Lucretia/tamp and follow the instructions for building gcc there, you'll find that I've already done the work to get a bare metal compiler going on ARM using a custom and very bare RTS without any nasty hacks in the makefiles.

[…]

# Conference Calendar

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

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

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

## 2012

July 01-03 — 24th **International Conference on Software Engineering and Knowledge Engineering** (SEKE'2012), Redwood City, California, USA. Topics include: Integrity, Security, and Fault Tolerance; Reliability; Component-Based Software Engineering; Embedded Software Engineering; Reverse Engineering; Programming Languages and Software Engineering; Program Understanding; Software Assurance; Software dependability; Software economics; Software Engineering Tools and Environments; Software Maintenance and Evolution; Software product lines; Software Quality; Software Reuse; Software Safety; Software Security; Software Engineering Case Study and Experience Reports; etc.

☺ July 09-11 — **GNU Tools Cauldron 2012**, Prague, Czech Republic. Sponsored by: AdaCore, Google, IBM. Topics include: gathering of GNU tools developers.

☺ July 10-13 — 10th IEEE **International Symposium on Parallel and Distributed Processing with Applications** (ISPA'2012), Madrid, Spain. Topics include: Parallel and Distributed Algorithms, and Applications; High-performance scientific and engineering computing; Middleware and tools; Reliability, fault tolerance, and security; Parallel/distributed system architectures; Tools/environments for parallel/distributed software development; Novel parallel programming paradigms; Compilers for parallel computers; Distributed systems and applications; etc.

☺ July 11-13 — 24th **Euromicro Conference on Real-Time Systems** (ECRTS'2012), Pisa, Italy. Topics include: avionics, aerospace, automotive applications; embedded devices; hardware/software co-design; compiler support; component-based approaches; middleware and distribution technologies; programming languages and operating systems; modelling and formal methods; etc.

July 16-20 — 36th Annual **International Computer Software and Applications Conference** (COMPSAC'2012), Izmir, Turkey. Topics include: Software life cycle, evolution, and maintenance; Formal methods; Software architecture and design; Reliability, metrics, and fault tolerance; Security; Real-time and embedded systems; Education and learning; Applications; etc.

July 18-20 — 17th Annual IEEE **International Conference on the Engineering of Complex Computer Systems** (ICECCS'2012), Paris, France. Topics include: Verification and validation, Model-driven development, Reverse engineering and refactoring, Design by contract, Agile methods, Safety-critical & fault-tolerant architectures, Real-time and embedded systems, Tools and tool integration, Industrial case studies, etc.

☺ August 27-28 — 17th International **Workshop on Formal Methods for Industrial Critical Systems** (FMICS'2012), Paris, France. Co-located with FM'2012. Topics include: Design, specification, code generation and testing based on formal methods; Methods, techniques and tools to support automated analysis, certification, debugging, learning, optimization and transformation of complex, distributed, real-time systems and embedded systems; Verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); Tools for the development of formal design descriptions; Case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; Impact of the adoption of formal methods on the development process and associated costs; Application of formal methods in standardization and industrial forums.

August 27-28 12th **International Conference on Quality Software** (QSIC'2012), Xi'an, China. Theme: "Engineering of Quality Software".

August 27-31 18th **International Symposium on Formal Methods** (FM'2012), Paris, France. Theme: "Interdisciplinary Formal Methods". Topics include: Interdisciplinary formal methods (techniques, tools and experiences demonstrating formal methods in interdisciplinary frameworks); Formal methods in practice (industrial applications of formal methods, experience with introducing formal methods in industry, tool usage reports, etc); Tools for formal methods (advances in automated verification and model-checking, integration of tools, environments for formal methods, etc); Role of formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, method integration, qualitative or quantitative improvements); Theoretical foundations (all aspects of theory related to specification, verification, refinement, and static and dynamic analysis); Teaching formal methods (original contributions that provide insight, courses of action regarding the teaching of formal methods, teaching experiences, educational resources, integration of formal methods into the curriculum, etc).

September 05-08 38th **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2012), Cesme, Izmir, Turkey. Topics include: information technology for software-intensive systems.

☺ Sep 05-08 **Track on Embedded Software Engineering** (ESE'2012). Topics include: Design and implementation of embedded software; Programming methodologies and languages for embedded software; Model-based and component-based approaches to embedded software development; Embedded software verification and validation; Testing and certification of embedded software; Software-intensive systems applications, e.g. in automotive, avionics, energy, industrial automation, health care, and telecommunication; Embedded software architectures; etc.

September 10-12 17th **European Symposium on Research in Computer Security** (ESORICS'2012), Pisa, Italy. Topics include: accountability, information hiding, information flow control, integrity, formal security methods, language-based security, risk analysis and management, security verification, software security, etc.

☺ September 10-13 41st **International Conference on Parallel Processing** (ICPP'2012), Pittsburgh, PA, USA. Topics include: all aspects of parallel and distributed computing, such as Architecture; Programming Models, Languages & Environments; Compilers and Run-Time Systems; Applications; etc.

September 10 5th **International Workshop on Parallel Programming Models and Systems Software for High-end Computing** (P2S2'2012).

September 13 **International Workshop on Embedded Multicore Systems** (EMS'2012). Topics include: Compilers for heterogeneous embedded multi-core systems; Programming models for embedded multi-core systems; Embedded OS designs and performance tuning tools; Formal methods for embedded systems; etc.

September 10-13 8th **International Conference on Open Source Systems** (OSS'2012), Hammamet, Tunisia. Theme: "Long-Term Sustainability with OSS". Deadline for early registration: August 10, 2012.

September 11-13 19th **International Static Analysis Symposium** (SAS'2012), Deauville, France. Topics include: abstract interpretation, bug detection, data flow analysis, model checking, new applications, program verification, security analysis, type checking, etc.

September 18-20 12th **International Workshop on Automated Verification of Critical Systems** (AVoCS'2012), Bamberg, Germany. Topics include: Specification and Refinement, Verification of Software and Hardware, Verification of Security-Critical Systems, Real-Time Systems, Dependable Systems, Verified System Development, Industrial Applications, etc. Deadline for submissions: July 23, 2012 (short papers). Deadline for registration: July 30, 2012.

September 19-20 6th **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2012), Lund, Sweden. Topics include: qualitative methods, empirical studies of software processes and products, industrial experience and case studies, evaluation and comparison of techniques and models, reports on the benefits / costs associated with using certain technologies, empirically-based decision making, quality measurement and assurance, software project experience and knowledge management, etc.

September 19-23 21st **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2012), Minneapolis, Minnesota, USA. Topics include: Parallel architectures and computational models;

Compilers and tools for parallel computer systems; Support for correctness in hardware and software (especially with concurrency); Parallel programming languages, algorithms and applications; Middleware and run time system support for parallel computing; Applications and experimental systems studies; etc.

September 23-30 28th IEEE **International Conference on Software Maintenance** (ICSM'2012), Riva del Garda, Trento, Italy. Topics include: reverse engineering and re-engineering, program and system comprehension, static and dynamic analysis, software migration and renovation, mining software repositories, maintenance and evolution processes, run-time evolution and update, empirical studies in software maintenance and evolution, testing in relation to maintenance (i.e., regression testing), etc.

September 25-28 5th **International Conference on Software Language Engineering** (SLE'2012), Dresden, Germany. Topics include: Formalisms used in designing and specifying languages and tools that analyze such language descriptions; Language implementation techniques; Program and model transformation tools; Language evolution; Approaches to elicitation, specification, or verification of requirements for software languages; Language development frameworks, methodologies, techniques, best practices, and tools for the broader language lifecycle; Design challenges in SLE; Applications of languages including innovative domain-specific languages or "little" languages; etc.

September 26-28 11th **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SoMeT'2012), Genoa, Italy. Topics include: software methodologies, and tools for robust, reliable, non-fragile software design; software developments techniques and legacy systems; software evolution techniques; agile software and lean methods; formal methods for software design; software maintenance; software security tools and techniques; formal techniques for software representation, software testing and validation; software reliability, and software diagnosis systems; Model Driven Development (DVD), code centric to model centric software engineering; etc.

October 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.

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 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. Deadline for submissions: July 9, 2012 (posters, ACM Student Research competition, Doctoral Symposium); July 11, 2012 (Dynamic Languages Symposium); July 15, 2012 (demonstrations), August, 2012 (workshop papers).

☺ 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. Deadline for submissions: August 10, 2012. Deadline for early registration: September 21, 2012.

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. Deadline for submissions: July-September, 2012 (workshop papers).

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 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 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. Deadline for submissions: July 18, 2012.

♦ Dec 02-06 ACM SIGAda Annual **International Conference on High Integrity Language Technology** (HILT'2012), Boston, Massachusetts, USA. Deadline for submissions: August 1, 2012 (industrial presentations).

☺ 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 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. Deadline for submissions: July 1, 2012 (workshops, tutorials), July 20, 2012 (papers).

☺ 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 submissions: September 16, 2012 (student symposium). 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. Deadline for submissions: July 6, 2012 (paper registration), July 10, 2012 (papers).

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. Deadline for submissions: September 30, 2012 (papers).

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. Deadline for submissions: July 30, 2012 (papers).

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. Deadline for submissions: September 21, 2012 (full papers)

☺ 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: September 21, 2012 (papers), October 31, 2012 (research abstracts).

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, ...). Deadline for submissions: July 23, 2012 (round 2), October 8, 2012 (round 3).

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: August 17, 2012 (technical research papers), 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).

♦ June 10-14 18th **International Conference on Reliable Software Technologies – Ada-Europe'2013**, Berlin, Germany. Deadline for submissions: December 3, 2012 (papers, tutorials, workshops), January 14, 2013 (industrial presentations).

☺ 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.

## ACM SIGAda Annual International Conference

# High Integrity Language Technology
# HILT 2012

## Preliminary Announcement / Call for Technical Contributions

### Developing and Certifying Critical Software

Hyatt Regency Boston
Boston, Massachusetts, USA
December 2-6, 2012

Sponsored by ACM SIGAda
SIGAda.HILT2012@acm.org
http://www.sigada.org/conf/hilt2012

### SUMMARY

*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 will provide a forum for experts from academia/research, industry, and government to present the latest findings in designing, implementing, and using language technology for high integrity software. To this end we are soliciting technical papers, experience reports (including experience in teaching), and tutorial proposals on a broad range of relevant topics.

### POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

- New developments in formal methods
- Multicore and high integrity systems
- Object-Oriented Programming in high integrity systems
- High-integrity languages (e.g., SPARK)
- Use of high reliability profiles such as Ravenscar
- Use of language subsets (e.g., MISRA C, MISRA C++)
- Software safety standards (e.g., DO-178B and DO-178C)
- Typed/Proof-Carrying Intermediate Languages
- Contract-based programming (e.g., Ada 2012)
- Model-based development for critical systems
- Specification languages (e.g., Z)
- Annotation languages (e.g., JML)
- Case studies of high integrity systems
- Analysis, testing, and validation
- Static and dynamic analysis of code
- System Architecture and Design including Service-Oriented Architecture and Agile Development
- Information Assurance
- Security and the Common Criteria / Common Evaluation Methodology
- Architecture design languages (e.g., AADL)
- Fault tolerance and recovery

### KEYNOTE SPEAKERS

HILT 2012 will feature internationally recognized experts:

**Kathleen Fisher**
DARPA

**Nancy Leveson**
MIT

**Barbara Liskov**
MIT

**Greg Morrisett**
Harvard Univ.

**Guy Steele**
Oracle Labs

## KINDS OF TECHNICAL CONTRIBUTIONS

***TECHNICAL ARTICLES*** present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM *Ada Letters*. The Proceedings will be entered into the widely consulted ACM Digital Library.

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

***EXPERIENCE REPORTS*** present timely results and "lessons learned". Submit a 1-2 page description of the project and the key points of interest, for publication in the proceedings. A paper will not be required.

***PANEL SESSIONS*** gather groups of experts to discuss/debate particular topics. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

***INDUSTRIAL PRESENTATIONS*** Submit a short overview (at least 1 page in length) of the proposed presentation. Authors of accepted presentations will be invited to submit full articles for ACM *Ada Letters*.

***WORKSHOPS*** are focused sessions on particular subjects. Workshop proposals, up to 5 pages in length, will be selected based on their applicability to the conference and potential for attracting participants.

***TUTORIALS*** can address a broad spectrum of topics relevant to the conference theme. Submissions will be evaluated based on applicability, suitability for presentation in tutorial format, and presenter's expertise. Tutorial proposals should include the expected level of experience of participants, an abstract or outline, the qualifications of the instructor(s), and the length of the tutorial (half day or full day).

***HOW TO SUBMIT***: Send in Word, PDF, or text format:

| *Submission* | *Deadline* | *Send to* |
|---|---|---|
| Technical articles, extended abstracts, experience reports, panel session proposals, or workshop proposals | **June 29, 2012** | Program Co-Chairs<br>Jeff Boleng – `jeff@boleng.com`<br>or<br>Tucker Taft – `taft@adacore.com` |
| Industrial presentation proposals | **August 1, 2012** (overview)<br>**October 1, 2012** (abstract) | |
| Tutorial proposals | **June 29, 2012** | John McCormick, Tutorials Chair<br>`mccormick@cs.uni.edu` |

At least one author for each accepted submission needs to register for and commit to presenting at the conference.

## FURTHER INFORMATION

***CONFERENCE GRANTS FOR EDUCATORS***: The ACM SIGAda Conference Grants program is designed to help educators introduce, strengthen, and expand the use of Ada and related technologies in school, college, and university curricula. The benefits include full conference and tutorial registration. For details please visit the conference web site or contact Prof. Michael B. Feldman (`MFeldman@gwu.edu`)

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

***SPONSORS AND EXHIBITORS***: Please contact Alok Srivastava (`asrivastava@yahoo.com`) to learn the benefits of becoming a sponsor and/or exhibitor at HILT 2012.

## ANY QUESTIONS?

Please send email to `SIGAda.HILT2012@acm.org`, or contact:

- HILT 2012 Conference Chair (Ben Brosgol, `brosgol@adacore.com`),
- SIGAda Vice-Chair for Meetings and Conferences (Alok Srivastava, `asrivastava@yahoo.com`), or
- SIGAda Chair (Ricky E. Sward, `rsward@mitre.org`).

# First 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

Förderverein Ada Deutschland e. V.

Special Interest Group Ada of the German Informatics Society (GI FG Ada)

**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 Languages for 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 Limited, 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
*Christoph Grein*, Ada Germany, Germany
*Peter Hermann*, Universität Stuttgart, Germany
*Jérôme Hugues*, ISAE Toulouse, France
*Pascal Leroy*, Google, Switzerland
*Albert Llemosí*, Universitat de les Illes Balears, Spain
*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*, Universitat Politècnica de València, Spain
*Jean-Pierre Rosen*, Adalog, France
*Ed Schonberg*, AdaCore, USA
*Tucker Taft*, AdaCore, USA
*Theodor Tempelmeier*, Univ. of Applied Sciences Rosenheim, Germany
*Tullio Vardanega*, Università di Padova, Italy

## Industrial Committee

*to be announced*

### 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=adaeurope2013*). 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 proposed presentation by January 14, 2013. Please follow the submission instructions on the conference website. The *Industrial Committee* will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by 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 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

ESTEREL Technologies

## Critical Systems and Software Development Solutions





**Esterel Technologies is the worldwide leader of model-based design, verification and code generation tools for critical system and software development.**

**Esterel Technologies' SCADE® product family** provides complete solutions that easily integrate, allowing for development optimization and increased communication among team members.

### SCADE Suite®
Control and Logic Application Development

### SCADE Display®
Display and HMI Development

### SCADE System™
System Architecture Design

### SCADE LifeCycle™
Application Lifecycle Management

### SCADE Solutions for ARINC 661 Compliant Systems
ARINC 661-compliant avionics displays

SCADE qualified/certified
code generators produce C and Ada

Visit www.esterel-technologies.com to learn more.

Esterel Technologies SA - 8, rue Blaise Pascal - Parc Euclide - 78990 Elancourt
Phone: +33 1 30 68 61 60 - sales@esterel-technologies.com

# Rationale for Ada 2012: 3 Structure and visibility

***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 areas of structure and visibility for Ada 2012.*

*Perhaps the most amazing change is that functions may now have parameters of all modes. In earlier versions of Ada, functions could only have parameters of mode* ***in*** *and so could not change variables explicitly passed as parameters; however, they could silently manipulate global variables in any way whatsoever. In order to ameliorate any risks of foolishness with this new freedom, there are new rules regarding order dependence.*

*There are also important improvements to incomplete types which make them much more useful; these include completion by a private type, their use as parameters and a new form of generic parameter.*

*Other improvements include a new form of use clause and changes to extended return statements.*

*Keywords: rationale, Ada 2012.*

## 1 Overview of changes

The WG9 guidance document [1] does not specifically identify problems in this area other than through a general exhortation to remedy shortcomings.

The following Ada Issues cover the relevant changes and are described in detail in this paper:

15 Constant return objects

19 Primitive subprograms are frozen with a tagged type

32 Extended return statements for class-wide functions

53 Aliased views of unaliased objects

142 Explicitly aliased parameters

143 In out parameters for functions

144 Detecting dangerous order dependencies

150 Use all type clause

151 Incomplete types as parameters and result

162 Incomplete types completed by partial views

213 Formal incomplete types

214 Default discriminants for limited tagged types

235 Accessibility of explicitly aliased parameters

277 Aliased views of extended return objects

296 Freezing of subprograms with incomplete parameters

These changes can be grouped as follows.

First there is the exciting business of allowing parameters of all modes for functions (143) and the associated rules to prevent certain order dependences (144). Another change concerning parameters is permitting explicitly aliased parameters (142, 235).

There are then a number of improvements in the area of incomplete types (151, 162) including the ability to permit them as formal generic parameters (213, 296). There are also related changes to the freezing rules (19).

There is also a minor change regarding discriminants (214).

The existing two forms of use clause (use package clause and use type clause) are augmented by a third form: the use all type clause (150).

Finally, there are a number of changes (corrections really) to extended return statements which were introduced in Ada 2005 (15, 32, 277). An associated change is the introduction of the idea of an immutably limited type (53).

## 2 Subprogram parameters

The main topic here is the fact that functions (but not operators) in Ada 2012 can have parameters of any mode.

This is a topic left over from Ada 2005. The epilogue to the Rationale for Ada 2005 [2] discusses a number of topics that were abandoned and in the case of function modes says:

"Clearly, Ada functions are indeed curious. But strangely this AI (that is AI95-323) was abandoned quite early in the revision process on the grounds that it was 'too late'. (Perhaps too late in this context meant 25 years too late.)" It was not possible to agree on a way forward and so effort was devoted to other topics.

But mists clear with time. The big concern was that allowing parameters of all modes might open the door to dangerous programming practices but a solution to that was found in the introduction of stricter rules preventing many order dependences.

It is instructive to quickly go through the various historical documents.

A probably little known document is one written in 1976 by David Fisher of the Institute for Defense Analyses [3] which provided the foundation for the requirements for the development of a new language. It doesn't seem to distinguish between procedures and functions; it does

mention the need for parameters which are constant and those which effectively rename a variable. Moreover, it does say (item C1 on page 81): *Side effects which are dependent on the evaluation order among the arguments of an expression will be evaluated left-to-right.* This does not actually require left-to-right evaluation but the behaviour must be as if it were. I have always thought it tragic that this was not observed.

This document was followed by a series known as Strawman, Woodenman, Tinman, Ironman [4] and finishing with Steelman [5].

The requirement on left-to-right evaluation remained in Tinman and was even stronger in Ironman but was somewhat weakened in Steelman to allow instrumentation and ends with a warning about being erroneous.

Further requirements are introduced in Ironman which requires both functions and procedures as we know them. Moreover, Ironman has a requirement about assignment to variables non-local to a function; they must be encapsulated in a region that has no calls on the function; this same requirement notes that it implies that functions can only have input parameters. This requirement does not seem to have carried forward to Steelman.

However, Ironman also introduces a requirement on restrictions to prevent aliasing. One is that the same actual parameter of a procedure cannot correspond to more than one input-output parameter. This requirement does survive into Steelman. But, it only seems to apply to procedures and not to functions and Steelman appears not to have noticed that the implied requirement that functions can only have input parameters has vanished.

It interesting to then see what was proposed in the sequence of languages leading to Ada 83, namely, Preliminary Green [6], Green [7], Preliminary Ada [8], and Ada [9]. Note that Preliminary Green was based on Ironman whereas Green was based on Steelman.

In Preliminary Green we find procedures and functions. Procedures can have parameters of three modes, **in**, **out** and **access** (don't get excited, **access** meant **in out**). Functions can only have parameters of mode **in**. Moreover,

> side effects to variables accessible at the function call are not allowed. In particular, variables that are global to the function body may not be updated in the function body. The rationale for Preliminary Green makes it quite clear that functions can have no side effects whatsoever.

In Green we find the three modes **in**, **out**, and **in out**. But the big difference is that as well as procedures and functions as in preliminary Green, there are now value returning procedures such as

```
procedure Random return Real range –1.0 .. 1.0;
```

The intent is that functions are still free of all side effects whereas value returning procedures have more flexibility. However, value returning procedures can only have parameters of mode **in** and

> assignments to global variables are permitted within value returning procedures. Calls of such procedures are only valid at points of the program where the corresponding variables are not within the scope of their declaration. The order of evaluation of these calls is strictly that given in the text of the program. Calls to value returning procedures are only allowed in assignment statements, initializations and procedure calls.

The rationale for Green notes that if you want to instrument a function then use a pragma. It also notes that functions

> with arbitrary side effects would undermine the advantage of the functional approach to software. In addition it would complicate the semantics of all language structures where expressions involving such calls may occur. Hence this form of function is not provided.

And now we come to Ada herself. There are manuals dated July 1979 (preliminary Ada), July 1980 (draft mil-std), July 1982 (proposed ANSI standard), and January 1983 (the ANSI standard usually known as Ada 83).

In Preliminary Ada, we have procedures, functions and value returning procedures exactly as in Green. Indeed, it seems that the only difference between Green and Preliminary Ada is that the name Green has been converted to Ada.

But the 1980 Ada manual omits value returning procedures and any mention of any restrictions on what you can do in a function. And by 1982 we find that we are warned that parameters can be evaluated in any order and so on.

The Rationale for Ada 83 [10] didn't finally emerge until 1986 and discusses briefly the reason for the change which is basically that benevolent side effects are important. It concludes by quoting from a paper regarding Algol 60 [11]

> The plain fact of the matter is (1) that side-effects are sometimes necessary, and (2) programmers who are irresponsible enough to introduce side-effects unnecessarily will soon lose the confidence of their colleagues and rightly so.

However, an interesting remark in the Rationale for Ada 83 in the light of the change in Ada 2012 is

> The only limitation imposed in Ada on functions is that the mode of all parameters must be **in**: it would not be logical to allow **in out** and **out** parameters for functions in a language that excludes nested assignments within an expression.

Hmm. That doesn't really seem to follow. Allowing assignments in expressions as in C is obnoxious and one of the sources of errors in C programs. It is not so much that permitting side-effects in expressions via functions is unwise but more that treating the result of an assignment as a value nested within an expression is confusing. Such nested constructions are naturally still excluded from Ada 2012 and so it is very unlikely that the change will be regretted.

Now we must turn to the question of order dependences. Primarily, to enable optimization, Ada does not define the order of evaluation of a number of constructions. These include

- the parameters in a subprogram or entry call,
- the operands of a binary operator,
- the destination and value in an assignment,
- the components in an aggregate,
- the index expressions in a multidimensional name,
- the expressions in a range,
- the barriers in a protected object,
- the guards in a select statement,
- the elaboration of library units.

The expressions involved in the above constructions can include function calls. Indeed, as AI-144 states "Arguably, Ada has selected the worst possible solution to evaluation order dependences (by not specifying an order of evaluation), it does not detect them in any way, and then says that if you depend upon one (even if by accident), your code will fail at some point in the future when your compiler changes. Something should be done about this."

It is far too late to do anything about specifying the order of evaluation so the approach taken is to prevent as much aliasing as possible since aliasing is an important cause of order of evaluation problems. Ada 2012 introduces rules for determining when two names are "known to denote the same object".

Thus they denote the same object if

- both names statically denote the same stand-alone object or parameter; or
- both names are selected components, their prefixes are known to denote the same object, and their selector names denote the same component.

and so on with similar rules for dereferences, indexed components and slices. There is also a rule about renaming so that if we have

```
C: Character renames S(5);
```

then C and S(5) are known to denote the same object. The index naturally has to be static.

A further step is to define when two names "are known to refer to the same object". This covers some cases of overlapping. Thus given a record R of type T with a component C, we say that R and R.C are known to refer to the same object. Similarly with an array A we say that A and A(K) are known to refer to the same object (K does not need to be static in this example).

Given these definitions we can now state the two basic restrictions.

The first concerns parameters of elementary types:

- For each name N that is passed as a parameter of mode **in out** or **out** to a call of a subprogram S, there is no other name among the other parameters of mode **in out** or **out** to that call of S that is known to denote the same object.

Roughly speaking this comes down to saying two or more parameters of mode **out** or **in out** of an elementary type cannot denote the same object. This applies to both functions and procedures.

This excludes the example given in the Introduction which was

```
procedure Do_It(Double, Triple: in out Integer) is
begin
  Double := Double * 2;
  Triple := Triple * 3;
end Do_It;
```

with

```
Var: Integer := 2;
...
Do_It(Var, Var);          -- illegal in Ada 2012
```

The key problem is that parameters of elementary types are always passed by copy and the order in which the parameters are copied back is not specified. Thus Var might end up with either the value of Double or the value of Triple.

The other restriction concerns constructions which have several constituents that can be evaluated in any order and can contain function calls. Basically it says:

- If a name N is passed as a parameter with mode **out** or **in out** to a function call that occurs in one of the constituents, then no other constituent can involve a name that is known to refer to the same object.

Constructions cover many situations such as aggregates, assignments, ranges and so on as listed earlier.

This rule excludes the other example in the Introduction, namely, the aggregate

```
(Var, F(Var))             -- illegal in Ada 2012
```

where F has an **in out** parameter.

The rule also excludes the assignment

```
Var := F(Var);            -- illegal
```

if the parameter of F has mode **in out**. Remember that the destination of an assignment can be evaluated before or after the expression. So if Var were an array element such as A(I) then the behaviour could vary according to the order. To encourage good practice, it is also forbidden even when Var is a stand-alone object.

Similarly, the procedure call

```
Proc(Var, F(Var));        -- illegal
```

is illegal if the parameter of F has mode **in out**. Examples of overlapping are also forbidden such as

```
ProcA(A, F(A(K));        -- illegal
ProcR(R, F(R.C));        -- illegal
```

assuming still that F has an **in out** parameter and that ProcA and ProcR have appropriate profiles because, as explained above, A and A(K) are known to refer to the same object as are R and R.C.

On the other hand

```
Proc(A(J), F(A(K));      -- OK
```

is permitted provided that J and K are different objects because this is only a problem if J and K happen to have the same value.

For more details the reader is referred to the AI. The intent is to detect situations that are clearly troublesome. Other situations that might be troublesome (such as if J and K happen to have the same value) are allowed, since to prevent them would make many programs illegal that are not actually dubious. This would cause incompatibilities and upset many users whose programs are perfectly correct.

The other change in Ada 2012 concerning parameters is that they may be explicitly marked **aliased** thus

```
procedure P(X: aliased in out T; ... );
```

As a consequence within P we can write X'Access. Recall that tagged types were always considered implicitly aliased anyway and always passed by reference. If the type T is a by-copy type such as Integer, then adding **aliased** causes it to be passed by reference. (So by-copy types are not always passed by copy!)

The possibility of permitting explicitly aliased function results such as

```
function F( ... ) return aliased T;      -- illegal Ada 2012
```

was considered but this led to difficulties and so was not pursued.

The syntax for parameter specification is modified thus

```
parameter_specification ::=
  defining_identifier_list: [aliased] mode [null exclusion]
                    subtype_mark [:= default_expression]
| defining_identifier_list: access_definition
                                        [:= default_expression]
```

showing that **aliased** comes first as it does in all contexts where it is permitted.

The rules for mode conformance are modified as expected. Two profiles are only mode conformant if both or neither are explicitly marked as aliased. Although adding **aliased** for a tagged type parameter makes little difference since tagged types are implicitly aliased, if this is done for a subprogram declaration then it must be done for the corresponding body as well.

There are (of course) rules regarding accessibility; these are much as expected although a special case arises in function return statements. As usual, if the foolish programmer does something silly, the compiler will draw attention to the error.

Explicitly aliased parameters were largely introduced to overcome problems in the container library. Examples will be given in the paper addressing containers.

## 3 Incomplete types

Incomplete types in Ada 83 were very incomplete. They were mostly used for the traditional linked list such as

```
type Cell;                          -- incomplete
type Cell_Ptr is access Cell;

type Cell is                        -- the completion
  record
    Next: Cell_Ptr;
    Element: Pointer;
  end record;
```

The incomplete type could only be used in the declaration of an access type. Moreover, the incomplete declaration and its completion had to be in the same list of declarations. However, if the incomplete declaration is in a private part then the completion can be deferred to the body; this is the so-called Taft Amendment added to Ada 83 at the last minute.

Ada 95 introduced tagged types and generalized access types and so made the language much more flexible but made no changes to incomplete types as such. However, it soon became clear that the restrictive nature of incomplete types was a burden regarding mutually dependent types and was a key issue in the requirements for Ada 2005.

The big step forward in Ada 2005 was the introduction of the limited with clause. This enables a package to have an incomplete view of a type in another package and solves many problems of mutually recursive types.

However, the overall rule remained that an incomplete type could only be completed by a full type declaration and, moreover, a parameter could not (generally) be of an incomplete type. This latter restriction encouraged the use of access parameters.

As mentioned in the Introduction, the first rule prevented the following

```
type T1;
type T2 (X: access T1) is private;
type T1 (X: access T2) is private;  -- illegal in Ada 2005
```

since the completion of T1 could not be by a private type.

This is changed in Ada 2012 so that an incomplete type can be completed by any type (other than another incomplete type). Note especially that an incomplete type can be completed by a private extension as well as by a private type.

The other major problem in Ada 2005 was that with mutually dependent types in different packages we could not use incomplete types as parameters because it was not known whether they were by-copy or by-reference. Of

course, if they were tagged then we did know they were by reference but that was a severe restriction.

The need to know whether parameters are by reference or by copy was really a red herring. The model used for parameter passing in versions of Ada up to and including Ada 2005 was basically that at the point of the declaration of a subprogram we need to have all the information required to call the subprogram. Thus we needed to know how to pass parameters and so whether they were by reference or by copy.

But this is quite unnecessary; we don't need to know the mechanisms involved until a point where the subprogram is actually called or the body itself is encountered since it is only at those points that the parameter mechanism is really required. It is only at those points that the compiler has to grind out the code for the call or for the body.

So the rules in Ada 2012 are changed to use this "when we need to know" model. This is discussed in AI-19 which is actually a binding interpretation and thus retrospectively applies to Ada 2005 as well. This is formally expressed by the difference between freezing a subprogram and freezing its profile. This was motivated by a problem with tagged types whose details need not concern us.

As a highly benevolent consequence, we are allowed to use incomplete types as both parameters and function results provided that they are fully defined at the point of call and at the point where the body is defined.

But another consequence of this approach is that we cannot defer the completion of an incomplete type declared in a private part to the corresponding body. In other words, parameters of an incomplete type are allowed provided the Taft Amendment is not used for completing the type.

The other exciting change regarding incomplete types is that in Ada 2012 they are allowed as generic parameters. In Ada 2005 the syntax is

```
formal_type_declaration ::=
  type defining_identifier [discriminant_part] is
                                        formal_type_definition ;
```

whereas in Ada 2012 we have

```
formal_type_declaration ::=
          formal_complete_type_declaration
        | formal_incomplete_type_declaration

formal_complete_type_declaration ::=
  type defining_identifier [discriminant_part] is
                                        formal_type_definition ;

formal_incomplete_type_declaration ::=
  type defining_identifier [discriminant_part] [is tagged] ;
```

So the new kind of formal generic parameter has exactly the same form as the declaration of an incomplete type. It can be simply **type** T; or can require that the actual be tagged by writing **type** T **is tagged**; – and in both cases a discriminant can be given.

A formal incomplete type can then be matched by any appropriate incomplete type. If the formal specifies **tagged**, then so must the actual. If the formal does not specify **tagged** then the actual might or might not be tagged. Of course, a formal incomplete type can also be matched by an appropriate complete type. And also, in all cases, any discriminants must match as well.

An example of the use of a formal incomplete type occurs in the package Ada.Iterator_Interfaces whose generic formal part is

```
generic
  type Cursor;
  with function Has_Element(Position: Cursor)
                                              return Boolean;
package Ada.Iterator_Interfaces is ...
```

The formal type Cursor is incomplete and can be matched by an actual incomplete type. The details of this package will be described in a later paper.

Another example is provided by a signature package as mentioned in the Introduction. We can write

```
generic
  type Element;
  type Set;
  with function Empty return Set is <>;
  with function Unit(E: Element) return Set is <>;
  with function Union(S, T: Set) return Set is <>;
  with function Intersection(S, T: Set) return Set is <>;
  ...
package Set_Signature is end;
```

Such a signature generic can be instantiated with an actual set type and then the instance can be passed into other generics that have a formal package such as

```
generic
  type VN is private;
  type VN_Set is private;
  with package Sets is
    new Set_Signature(Element => VN, Set => VN_Set,
                                              others => <>);
  ...
package Analyse is ...
```

This allows the construction of a generic that needs a Set abstraction such as a flow analysis package. Remember that the purpose of a signature is to group several entities together and to check that various relationships hold between the entities. In this case the relationships are that the types Set and Element do have the various operations Empty, Unit and so on.

The set generic could be included in a set container package thus

```
generic
  type Element is private;
package My_Sets is
  type Set is tagged private;

  function Empty return Set;
```

```
   function Unit(E: Element) return Set;
   function Union(S, T: Set) return Set;
   function Intersection(S, T: Set) return Set;
   ...
   package My_Set is new Set_Signature(Element, Set);
private
   ...
end My_Sets;
```

The key point is that normally an instantiation freezes a type passed as a generic parameter. But in the case of a formal incomplete untagged type, this does not happen. Hence the actual in the instantiation of Set_Signature in the generic package My_Sets can be a private type such as Set.

This echoes back to the earlier discussion of changing the freezing rules. We cannot call a subprogram with untagged incomplete parameters (whether formal or not) because we do not know whether they are to be passed by copy or by reference. But we can call a subprogram with tagged incomplete parameters because we do know that they are passed by reference (and this has to remain true for compatibility with Ada 2005). So just in case the actual subprogram in the tagged case is called within the generic, the instantiation freezes the profile. But in the untagged case, we know that the subprogram cannot be called and so there is no need to freeze the profile.

This means that the type Set should not be given as tagged incomplete in the package Set_Signature since we could not then use the signature in the package My_Sets.

If a subprogram has both tagged and untagged formal incomplete parameters then the untagged incomplete parameters win and the subprogram cannot be called.

(If this is all too confusing, do not worry, the compiler will moan at you if you make a mistake.)

Another rule regarding incomplete formal types is that the controlling type of a formal abstract subprogram cannot be incomplete.

## 4 Discriminants

There is one minor change in this area which was mentioned in the Introduction.

In Ada 2005, a discriminant can only have a default if it is not tagged. But in Ada 2012, a default is also permitted in the case of a limited tagged type.

Ada typically uses defaults as a convenience so that in many cases standard information can be omitted. Thus it is convenient that the procedure New_Line has a default of 1 since it would be boring to have to write New_Line(1); all the time.

In the case of discriminants however, a default as in

```
type Polynomial(N: Index := 0) is
   record
      A: Integer_Vector(0 .. N);
   end record;
```

also indicates that the type is mutable.

However, tagged types in Ada 2005 never have defaults because we do not want tagged types to be mutable. On the other hand if a tagged type is limited then it is immutable anyway. And so it was concluded that there is no harm in permitting a limited tagged type to have a default discriminant.

This may seem rather academic but the problem arose in designing containers for queues. It was felt desirable that the protected type Queue should have a discriminant giving its ceiling priority and that this should have a default for convenience. As illustrated in the Introduction this resulted in a structure as follows

```
generic
   with package Queue_Interfaces is new ...
   Default_Ceiling: Any_Priority := Priority'Last;
package AC.Unbounded_Synchronized_Queues is
   ...
   protected type Queue(Ceiling: Any_Priority :=
                                      Default_Ceiling)
            with Priority => Ceiling
          is new Queue_Interfaces.Queue with ...
```

Now the problem is that a protected type such as Queue which is derived from an interface is considered to be tagged because interfaces are tagged. On the other hand a protected type is always limited and its discriminant provides a convenient way of providing the ceiling priority. So there was a genuine need for a change to the rule.

Note incidentally that the default is itself provided with the default value of Priority'Last since it is a generic parameter with its own default.

## 5 Use clauses

Ada 2012 introduces a further form of use clause. In order to understand the benefit it is perhaps worth just recalling the background to this topic.

The original use clause in Ada 83 made everything in a package directly visible. Consider the following package

```
package P is
   I, J, K: Integer;

   type Colour is (Red, Orange, Yellow, Green, Blue, ... );
   function Mix(This, That: Colour) return Colour;

   type Complex is
      record
         RI, Im: Float;
      end record;
   function "+"(Left, Right: Complex) return Complex;
   ...
end P;
```

Now suppose we have a package Q which manipulates entities declared in P. We need a with clause for P, thus

```
with P;
package Q is ...
```

With just a with clause for P we have to refer to entities in P using the prefix P. So we get statements and declarations in Q such as

```
P.I := P.J + P.K;

Mucky: P.Colour := P.Mix(P.Red, P.Green);

W: P.Complex := (1.0, 2.0);
Z: P.Complex := (4.0, 5.0);
D: P.Complex := P."+"(W, Z);
```

This is generally considered tedious especially if the package name is not P but A_Very_Long_Name. However, adding a package use clause to Q thus

```
with P; use P;
package Q...
```

enables the P prefix to be omitted and in particular allows infix notation for operators so we can now simply write

```
D: Complex := W + Z;
```

But as is well known, the universal use of such use clauses introduces ambiguity (if the same identifier is in two different packages and we have a use clause for both), obscurity (you can't find the wretched declaration of Red) and possibly a maintenance headache (another package is added which duplicates some identifiers). So there is a school of thought that use clauses are bad for you.

However, although the prefix denoting the package is generally beneficial it is a pain to be forced to always use the prefix notation for operators. So in Ada 95, the use type clause was added enabling us to write

```
with P; use type P.Complex;
package Q is ...
```

This has the effect that only the primitive *operators* of the type Complex are directly visible. So we can now write

```
D: P.Complex := W + Z;
```

Note that the type name Complex is not itself directly visible so we still have to write P.Complex in the declaration of D.

However, some users still grumbled. Why should only those primitive operations that happen to be denoted by operators be visible? Why indeed? Why cannot Mucky be declared similarly without using the prefix P for Mix, Red and Green?

It might be worth briefly recalling exactly which operations of a type T are primitive operations of T. They are basically

- predefined operations such as "=" and "+",
- subprograms declared in the same package as T and which operate on T,
- enumeration literals of T,
- for a derived type, inherited or overridden subprograms.

The irritation is solved in Ada 2012 by the **use all type** clause which makes all primitive operations visible. (Note another use for the reserved word **all**.)

So we can write

```
with P; use all type P.Colour;
package Q is ...
```

and now within Q we can write

```
Mucky: P.Colour := Mix(Red, Green);
```

Thus the enumeration literals such as Red are made directly visible as well as obvious primitive subprograms such as Mix.

Another impact concerns tagged types and in particular operations on class wide types.

Remember that subprograms with a parameter (or result) of type T'Class are not primitive operations unless they also have a parameter (or result of type T) as well.

Actually it is usually very convenient that operations on a class wide type are not primitive operations because it means that they are not inherited and so cannot be overridden. Thus we are assured that they do apply to all types of the class.

So, suppose we have

```
package P is
  type T is tagged private;
  procedure Op1(X: in out T);
  procedure Op2(Y: in T; Z: out T);
  function Fop(W: T) return Integer;
  procedure List(TC: in T'Class);
private
  ...
end P;
```

Then although List is not a primitive operation of T it will certainly look to many users that it belongs to T in some broad sense. Accordingly, writing **use all type** P.T; makes not only the primitive operations such as Op1, Op2 and Fop, visible but it also makes List visible as well.

Note that this is the same as the rule regarding the prefixed form of subprogram calls which can also be used for both primitive operations and class wide operations. Thus given an object A of type T, as well as statements A.Op1; and A.Op2(B); and a function call A.Fop we can equally write

```
A.List;          -- prefixed call of class wide procedure
```

Moreover, suppose we declare a type NT in a package NP thus

```
package NP is
  type NT is new T with ...
  ...
end NP;
```

If we have an object AN of type NT then not only can we use prefixed calls for inherited and overridden operations but we can also use prefixed calls for class wide operations in ancestor packages such as P. So we can write

```
AN.List;         -- prefixed call of List in ancestor package
```

Similarly, writing **use all type** NP.NT; on Q makes the inherited (or overridden) operations Op1, Op2 and Fop visible and also makes the class wide operation List declared in P visible. We do not also have to write **use all type** P.T; on Q as well.

We conclude by remarking that the maintenance problem of name clashes really only applies to use package clauses. In the case of use type and use all type clauses, the entities made visible are overloadable and a clash only occurs if two have the same profile which is very unlikely and almost inevitably indicates a bug.

# 6 Extended return statements

The final topic in this paper is the extended return statement. This was introduced in Ada 2005 largely to solve problems with limited types. However, some glitches have come to light and these are corrected in Ada 2012.

A description of the reasons for and general properties of the extended return statement will be found in [2].

The syntax for extended return statement in Ada 2005 as found in [12] is

```
extended_return_statement ::=
  return defining_identifier: [aliased]
              return_subtype_indication {:= expression] [do
    handled_sequence_of_statements
  end return] ;
```

Before going further, it should be mentioned that there was some confusion regarding limited types and so the term immutably limited was introduced in the course of the maintenance of Ada 2005. There were various problems. Basically, limitedness is a property of a view of a type. Thus even in Ada 83 a private type might be limited but the full view found in the private part would not be limited. Ada 95 introduced explicitly limited types. Ada 2005 introduced coextensions and these could even include such obviously limited things as task types thus adding a limited part to what was otherwise a seemingly nonlimited type. It became clear that it was necessary to introduce a term which meant that a type was really and truly limited and could not subsequently become nonlimited for example in a private part or in a child unit. So a type is immutably limited if

- it is an explicitly limited record type,
- it is a task type, protected type or synchronized interface,
- it is a non-formal limited private type that is tagged or has an access discriminant with a default expression,
- it is derived from an immutably limited type.

It was then realised that there were problems with extended return statements containing an explicit **aliased**. Consequently, it was decided that there was really no need for **aliased** if there was a rule that immutably limited return objects were implicitly aliased. So **aliased** was removed from the syntax. However, some users had already written **aliased** and this would have introduced an irritating incompatibility. So finally it was decided that **aliased** could be written but only if the type were immutably limited.

Another small problem concerned constants. Thus we might write

```
return X: T do
   ...                          -- compute X
end return;
```

However, especially in the case of a limited type LT, we might also give the return object an initial value, thus

```
return X: LT := (A, B, C) do
   ...                          -- other stuff
end return;
```

Now it might be that although the type as a whole is limited one or more of its components might not be and so could be manipulated in the sequence of statements. But if we want to ensure that this does not happen, it would be appropriate to indicate that X was constant. But, almost surely by an oversight, we cannot do that since it is not permitted by the syntax. So the syntax needed changing to permit the addition of constant.

To aid the description the syntax in Ada 2012 is actually written as two productions as follows

```
extended_return_object_declaration ::=
     defining_identifier: [aliased] [constant]
                  return_subtype_indication {:= expression]

extended_return_statement ::=
  return extended_return_object_declaration [do
    handled_sequence_of_statements
  end return] ;
```

The other small change to the extended return statement concerns the subtype give in the profile of the function and that in the extended return statement itself. The result type of the function can be constrained or unconstrained but that given in the extended return statement must be constrained.

This can be illustrated by considering array types. (These examples are from [2].) Suppose we have

```
type UA is array (Integer range <>) of Float;
subtype CA is UA(1 .. 10);
```

then we can write

```
function Make( ... ) return CA is
begin
   ...
   return R: UA(1 .. 10) do          -- statically matches
      ...
   end return;
end Make;
```

This is allowed because the subtypes statically match.

If the subtype in the function profile is unconstrained then the result must be constrained either by its subtype or by its initial value. For example

```
function Make( ... ) return UA is
begin
  ...
  return R: UA(1 .. N) do
    ...
  end return;
end Make;
```

and here the result R is constrained by its subtype. A similar situation can arise with records with discriminants. Thus we can have

```
type Person(Sex: Gender) is ... ;

function F( ... ) return Person is
begin
  if ... then
    return R: Person(Sex => Male) do
      ...
    end return;
  else
    return R: Person(Sex => Female) do
      ...
    end return;
  end if;
end F;
```

which shows that we have the possibility of returning a person of either gender.

However, what is missing from Ada 2005 is that we can have analogous situations with tagged types in that a function might wish to return a value of some type in a class.

So we would like to write things such as

```
function F( ... ) return Object'Class is
begin
  if ... then
    return C: Circle do
      ...
    end return;
  elsif ... then
    return S: Square do
      ...
    end return;
   end if;
end F;
```

This is not permitted in Ada 2005 which required the types to be the same. This can be overcome by writing

```
return C: Object'Class := Circle_Func do
  ...
end return;
```

where Circle_Func is some local function that returns the required object of type Circle.

This is all rather irksome so the wording is changed in Ada 2012 to say that in this situation the subtype in the extended return statement need not be the same as that in the profile but simply must be covered by it. There are also related slight changes to the accessibility rules.

## References


[1] ISO/IEC JTC1/SC22/WG9 N412 (2002) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of the Amendment*.

[2] John Barnes (2008) *Ada 2005 Rationale*, LNCS 5020, Springer-Verlag.

[3] David Fisher (1976) *A Common Programming Language for the Department of Defense – Background and Technical Requirements*, Institute for Defense Analyses, Arlington, Virginia.

[4] Defense Advanced Research Projects Agency (1977) *Department of Defense Requirements for High Order Computer Programming Languages – Revised IRONMAN*, USDoD.

[5] Defense Advanced Research Projects Agency (1978) *Department of Defense Requirements for High Order Computer Programming Languages – STEELMAN*, USDoD.

[6] Jean Ichbiah et al (1978) *Preliminary Reference Manual for the Green Programming Language*, Honeywell Inc.

[7] Jean Ichbiah et al (1979) *Reference Manual for the Green Programming Language*, Honeywell Inc.

[8] ACM (1979) *Preliminary Ada Reference Manual*, SIGPLAN Notices, Vol 14, No 6.

[9] ANSI / Mil–Std 1815A (1983) *Ada Reference Manual.*

[10] Jean Ichbiah, John Barnes, Robert Firth, Mike Woodger (1986) *Rationale for the Design of the Ada Programming Language*, Honeywell & Alsys.

[11] B Higman (1963) *What everybody should know about Algol*, Computer Journal, vol 6, no 1, pp 50-56.

[12] S. T. Taft et al (eds) (2007) *Ada 2005 Reference Manual*, LNCS 4348, Springer-Verlag.

# Use of Model Driven Code Generation on the ASIM Project

***Steen Palm***

*Terma A/S, Vasekær 12, 2730 Herlev, Denmark; Tel: +45 4594 9665; email: sup@terma.com*

## Abstract

*This paper describes the approach used for the development of software controlling two instruments on a payload that will be placed on the International Space Station. The development approach is based on the principles of SAVOIR and ASSERT. For each of the two instruments, a UML component design is constructed where interfaces are decorated with stereotypes like ‹‹cyclic›› and ‹‹sporadic›› defining their concurrency behaviour. From the design of an instrument application (the Interface View) and Ada packages implementing the sequential behaviour of the interfaces (the Functional View), a final Ravenscar compliant implementation of the instrument software (the Concurrency View) is automatically generated.*

*Keywords: SAVOIR, ASSERT, TASTE, ASIM, UML, Ada.*

## 1 Introduction

Terma has developed mission-critical software for several satellites and manned space missions. The software has been used for numerous mission-critical tasks such as control of the European Robotic Arm (ERA) on the International Space Station and attitude and orbit control of the satellites Ørsted, SAC-C, Herschel and Planck.

On the ASIM project, Terma is responsible for the development of the software to control two instruments: MXGS and MMIA. Terma is Prime on the project and is consequently also responsible for deciding on methods and tools to be used on the project. We have utilized this unique opportunity to select new and promising technologies that we have worked with in ESA (European Space Agency) technology development projects. More specifically, we have picked SAVOIR FAIRE [1] and ASSERT [2] as the foundation for the design and implementation of the software controlling the two instruments.

*ASIM* (Atmosphere-Space Interactions Monitor) is a payload that will be placed on the Columbus module of the International Space Station (ISS). ASIM consists of:

- 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
- MXGS (Modular X and Gamma ray Sensor) will observe the terrestrial gamma flashes associated with severe thunderstorms
- Supporting Systems for Data Handling and Power Distribution

### 1.1 SAVOIR FAIRE

*SAVOIR* (Space Avionics Open Interface aRchitecture) is an initiative to improve the way that the European space community builds avionics subsystems. *SAVOIR FAIRE* (SAVOIR Fair Architecture and Interface Reference Elaboration) is an industrial working group, who is working towards the definition of a reference architecture for software on-board spacecraft platforms. The reference architecture is intended to facilitate specification of building blocks that can be developed, qualified and composed into compliant avionics software systems. Terma is member of the SAVOIR FAIRE working group and has also participated in a number of projects under the ESA technology development programmes aiming at improving and validating the proposed software reference architecture. The architectural principles are strongly influenced by the outcome of the ASSERT project.

### 1.2 ASSERT

The *ASSERT* (Automated proof-based System and Software Engineering for Real-Time systems) project was an integrated project partially funded by the European Commission and coordinated by ESA. The ASSERT consortium included Terma as well as 27 other partners representing the space industry, research laboratories, software houses and tool developers. The project started in September 2004 and ended in December 2007. 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.

ASSERT describes the system from different perspectives (called views), thereby supporting separation of concerns. For instance, the designer can concentrate entirely on the sequential behaviour of operations when defining the Functional View without being concerned with concurrency aspects, which are dealt with in the Interface View. When using ASSERT, a platform independent

component model is constructed containing the following views:

- *The Functional View*: Specifies the functional services provided by components and expresses their sequential behaviour.
- *The Interface View*: Characterizes the provided and required services of components and declares their intended concurrent behaviour. In this view, a provided service can be specified to execute, for example, as a cyclic operation.
- *The Deployment View*: Specifies the physical architecture of the system and the way in which components are to be deployed on it.

From the platform independent specification, a platform dependent model is automatically constructed (by vertical transformation) containing the single view:

- *The Concurrency View:* Specifies the concurrent architecture of the system needed to implement the platform independent specification of it; this view is designed to be compliant with the Ravenscar Computational Model by construction.

### 1.3 TASTE

A tool chain called *TASTE* [3] (The Assert Set of Tools for Engineering) supporting the ASSERT process has been developed by ESA, together with a set of partners from the space industry. TASTE allows software designers to integrate heterogeneous pieces of code produced either manually (in C or Ada) or automatically by external modelling tools such as Matlab Simulink, SCADE, or Real-Time Developer Studio. Consistency of the integration is ensured through the use of two formal modelling and specification languages: AADL and ASN.1. Terma has applied TASTE on a number of ESA technology development projects.

## 2 The Terma Modelling Tool Chain

### 2.1 Overview

TASTE was the top candidate as the tool chain to be selected for the development of the instrument software on ASIM. However, TASTE had to be abandoned for the following reasons:

- TASTE does not support the target platform (GNAT Pro for LEON3 Bare Board with GNAT Ravenscar run-time libraries).
- It is not possible to model communication with device drivers in TASTE, e.g. no support for modelling interrupt handling.

Therefore, Terma has developed its own modelling tool chain based on SAVOIR/ASSERT principles, which will secure a Ravenscar compliant implementation:

- The design will be expressed as a component model in UML. Components may be composite or simple (leaf components). Enterprise Architect from Sparx Systems has been selected as UML tool.
- The Interface View is supported by specific UML stereotypes (like ‹‹protected›› and ‹‹cyclic››) that can be used to decorate interfaces provided by model components. The stereotypes are complemented with real-time attributes like worst-case execution time and period, which are supplied as UML tagged values associated with the provided interfaces.

  Compared to TASTE, the Interface View has been extended with stereotypes supporting interaction with device drivers (e.g. ‹‹interrupt sporadic››). Extensions have been introduced in a way that ensures that the generated Concurrency View is Ravenscar compliant.
- The Functional View is implemented as passive Ada packages. For each leaf component in the design, there will be a corresponding Ada package that defines the sequential behaviour of the operations provided by the component.
- The Deployment View is not supported. As the instrument software is running on a single node, a Deployment View was considered superfluous.
- The Concurrency View (obtained by vertical transformation) is automatically generated from the UML design (more precisely, from an exported XMI file). The Concurrency View consists of Ada tasks and protected objects, which will call the passive Ada subprograms defined in the Functional View. In fact, the Concurrency View constitutes the final implementation.

Ada was chosen as implementation language for the following reasons:

- The stereotypes (like ‹‹cyclic›› and ‹‹protected››) of operation interfaces can easily be transformed to Ada built-in language constructs (like tasks and protected objects).
- The vertical transformation to the Concurrency View is operating system independent. The generated Ada packages only contain standard Ada language constructs.
- The Ravenscar profile is defined as a restricted usage of Ada features, thus making Ada the natural choice. Also, Ada compilers can check an Ada program for compliance with the profile.

### 2.2 The Technical Details

This paper will focus entirely on the Terma modelling tool chain and the associated software development process. The actual designs for the instrument software will not be presented. Instead, consider the design in Figure 1 of a toy system called Micro OBOSS. The name has been chosen to indicate that the toy system is a micro version of the Terma development framework called OBOSS (On-Board

Operations Support Software). Micro OBOSS consists of three main components: The *Spacelink* component receives telecommand packets (TCs) from the environment and sends back telemetry packets (TMs). The *Router* component routes received TCs to application processes based on destination information in the TCs. In the example, only a single application process exists, namely the *Display_Manager* component. More specifically, Micro OBOSS behaves as follows (starting from the upper left corner of the figure):

- The blocking sporadic operation *Deliver_TC* provided by the component *Space_Uplink* calls the required (blocking) interface *Get_Line*[1] and waits for the user to supply a name string. When a name string is supplied, *Deliver_TC* will construct a TC destined for the *Display_Manager* and forward the TC to the *Router* component by calling the required interface *Forward_TC*
- Via the interface connectors, the call of *Forward_TC* will lead to a call of the sporadic operation *Receicve_TC* provided by the *TC_Router* component. This operation will determine the destination for the TC (always the *Display_Manager* in this example) and pass on the TC to the destination by calling the required interface *Handle_Display_TC*
- Via the interface connectors, the call of *Handle_Display_TC* will lead to a call of the passive operation *Handle_TC* provided by the *TC_Handler* component. This operation will save the name included in the TC by calling the required interface *Set_Name*
- Every half second, the cyclic operation *Display* provided by the *Displayer* component will retrieve the saved name by calling the required interface *Get_Name* and check if the saved name is non-empty. If so, it will send the string "Hello <name>" as a TM to the *Router* component by calling the required interface *Forward_TM*
- Via the interface connectors, the call of *Forward_TM* will lead to a call of the passive operation *Receive_TM* provided by the *TM_Router* component. This operation will pass on the TM to the *Space_Downlink* component by calling the required interface *Forward_TM*
- Finally, the passive *Receive_TM* operation provided by the *Space_Downlink* component will output the TM (a string) on standard out by calling the required (hardware) interface *Put_Line* [1]

[1] The subprograms *Get_Line* and *Put_Line* belong to the Ada standard package *Text_IO*.

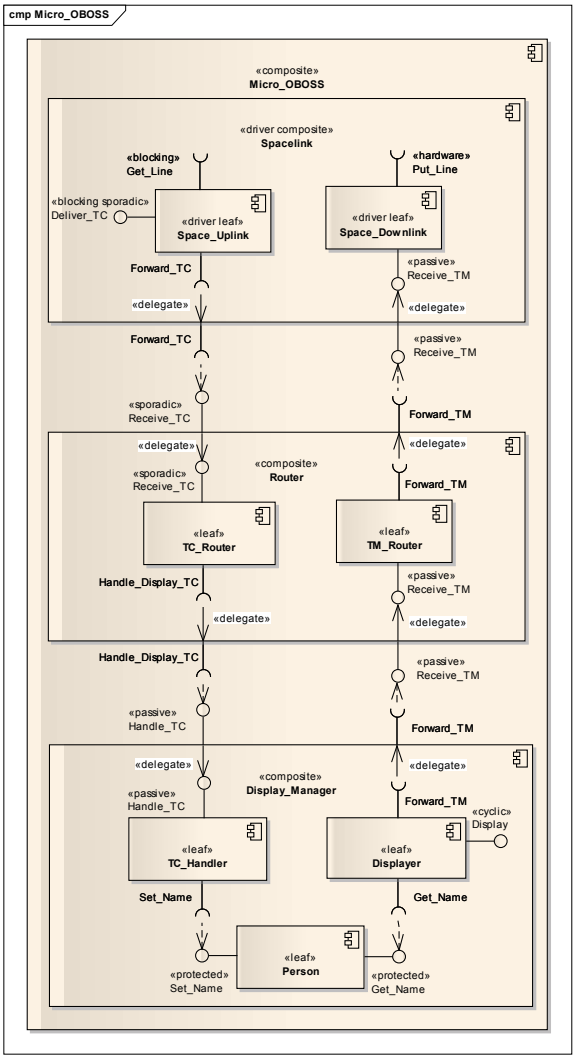


**Figure 1 Micro OBOSS example**

### *The Interface View*

The UML model in Figure 1 constitutes the Interface View. It consists of the following elements:

- ***Components***: Components may be composite or simple (leaf components). Only leaf components can perform functions. Composite components merely serve as containers supporting the structuring of the system to be designed. Components are further divided into regular components (with stereotypes ‹‹composite›› and ‹‹leaf››) and driver components (with stereotypes ‹‹driver composite›› and ‹‹driver leaf››). Driver components are intended to interface with the environment, for instance for interacting with hardware drivers. Blocking sporadic operations and interrupt sporadic operations (see below) must only be provided by driver components.
- ***Interfaces***: Two types of interfaces exist: Provided interfaces ( —○ ) and required interfaces ( —( ). The provided interfaces constitute the services provided by a component to other components,

whilst the required interfaces represent the services needed to implement the provided interfaces. Interfaces are further described below.

- ***Connectors***: Two types of connectors exist: Dependency connectors and delegate connectors.

  Dependency connectors are used to connect required interfaces to provided interfaces in order to provide the functionality required by a component.

  Delegate connectors are used to show how the provided functionality of a composite component is delegated to its constituent components and how the required functionality of its constituent components is delegated to the required interfaces of the composite component.

The UML model does not explicitly identify concurrency constructs such as periodic tasks or sporadic tasks, but focuses on the functional behaviour of the provided interfaces and thereby the functional behaviour of components. However, provided interfaces have stereotypes uniquely specifying their concurrency behaviour. These stereotypes thus render the need for tasking components in the design superfluous. The translation from stereotypes to actual Ada tasks and protected objects is done by the automatic transformation into the Concurrency View.

Table 1 defines the stereotypes of provided interfaces.

The provided interfaces also have a number of real-time attributes associated, such as WCET (worst-case execution time), MIAT (minimal inter-arrival time), Deadline, Queue size (number of requests that a sporadic operation must be able to store during a burst of requests), Period, and Interrupt ID, as shown in the table. These attributes must all be supplied by the designer.

Based on the stereotypes and the associated real-time attributes, the Terma modelling tool chain can perform a schedulability analysis of the system. The analysis is a so-called response time analysis, which calculates the worst-case response time for each task and compares it with the deadline for the task; see for instance [4].

Note that ‹‹cyclic›› and ‹‹blocking sporadic›› operations cannot be called. A ‹‹cyclic›› operation is automatically triggered when its period expires, while a ‹‹blocking sporadic›› operation is triggered when the called ‹‹blocking›› operation returns.

The required interfaces of a component collectively define the functionality required by the component in order for the component to provide the guaranteed functionality.

Required interfaces on regular components have no stereotypes. The reason is that a required interface specifies needed functionality and this is not dependent on the stereotype of the interface.

| Stereotype | Meaning | Attributes |
|---|---|---|
| ‹‹sporadic›› | An operation performed at sporadic intervals. Its appearance on a ‹‹leaf›› component specifies that the component contains a separate thread of execution. | WCET<br>MIAT<br>Deadline<br>Queue size |
| ‹‹cyclic›› | An operation performed at fixed intervals. Its appearance on a ‹‹leaf›› component specifies that the component contains a separate thread of execution. | WCET<br>Period<br>Deadline |
| ‹‹protected›› | An operation granted exclusive access to some state. | WCET |
| ‹‹passive›› | A regular operation | - |
| ‹‹interrupt sporadic›› | An interrupt service routine. An ‹‹interrupt sporadic›› operation must only be provided by a driver component. | WCET<br>MIAT<br>Deadline<br>Interrupt ID |
| ‹‹blocking sporadic›› | An operation that repeatedly handles data from the environment that it gets by calling a ‹‹blocking›› required interface. Its appearance on a ‹‹leaf›› component specifies that the component contains a separate thread of execution. A ‹‹blocking sporadic›› operation must only be provided by a driver component. | WCET<br>MIAT<br>Deadline |

**Table 1 Stereotypes of provided interfaces**

On driver components, the ‹‹hardware›› required interface and the ‹‹blocking›› required interface can be used. These two interface types both represent interaction with the environment. As opposed to all other required interfaces, ‹‹hardware›› and ‹‹blocking›› required interfaces shall not be connected to other interfaces (see *Get_Line* and *Put_Line* in Figure 1).

The ‹‹hardware›› required interface simply denotes the use of a regular subprogram provided by the environment, e.g. the reading of a hardware register.

The ‹‹blocking›› required interface denotes the call of a blocking subprogram provided by the environment, i.e. a subprogram that hangs until it has data to return to the caller (the procedure *Text_IO.Get_Line* is an example of such a subprogram). A component with a ‹‹blocking›› required interface must also have a ‹‹blocking sporadic›› provided interface (and vice versa).

Enterprise Architect allows the designer to specify signatures for the provided and required interfaces in the model. However, these signatures are Java/C specific and cannot be used to specify signatures for Ada subprograms.

Therefore, it has been decided to abandon signatures in the model and instead create two Ada specifications for each leaf component: One that defines the provided interfaces and one that defines the required interfaces. As an example consider the two Ada specifications for the *TC_Router* component (compare with Figure 1).

Ada specification for the provided interfaces of *TC_Router*:

```
with Data_View;
package Micro_OBOSS.Router.TC_Router is
   procedure Receive_TC(TC : in Data_View.TC_T);
end Micro_OBOSS.Router.TC_Router;
```

Ada specification for the required interfaces of *TC_Router*:

```
with Data_View;
package Micro_OBOSS.Router.TC_Router_Required
   is procedure Handle_Display_TC(
                                TC : in Data_View.TC_T);
end Micro_OBOSS.Router.TC_Router_Required;
```

### *The Functional View*

The functional view defines the sequential behaviour of the provided interfaces. This implies, that for each leaf component, an Ada body must be supplied that implements the Ada specification for the provided interfaces of the component by using the Ada specification for the required interfaces of the component. As an example consider the Ada body for the *TC_Router* component:

```
with Micro_OBOSS.Router.TC_Router_Required;
package body Micro_OBOSS.Router.TC_Router is
  package RI renames
        Micro_OBOSS.Router.TC_Router_Required;
  procedure Receive_TC(TC : in Data_View.TC_T) is
  begin
    case TC.Destination is
      when Data_View.DISPLAYER =>
        RI.Handle_Display_TC(TC);
      when others =>
        null;
    end case;
  end Receive_TC;
end Micro_OBOSS.Router.TC_Router;
```

The implementation must be completely independent of the connectors and only use the required interfaces (*Handle_Display_TC* in the example) to achieve its goal [2].

Although, the designer only provides implementations for leaf components, the Functional View is in essence an Ada package hierarchy, which is structured in exactly the same way as the UML model. The parent packages are automatically created by the vertical transformation into the Concurrency View in order to relieve the designer from this trivial task.

### *The Concurrency View*

The designer develops the Interface View and the Functional View of the system. The Concurrency View is then automatically generated from these two views. The Concurrency View constitutes the final implementation of the system.

Before the vertical transformation can be initiated, the UML design must be exported to an XMI 2.1 file [3]. The vertical transformation is implemented as an XSLT stylesheet that transforms the exported XMI file into the Concurrency View Ada package hierarchy.

The vertical transformation will perform the following:

- Create the Ada specifications in the Functional View Ada package hierarchy for composite components
- Determine the deadline monotonic priority-ordering between tasks (including interrupt routines) and save the result to a file. In case the designer wants a different priority ordering, (s)he can simply re-order the tasks within the file.
- Determine an optimal priority ordering between all tasks and protected objects in the system based on the task priority-ordering and the call pattern between tasks and protected operations (given by the connectors in the UML model).
- Create an Ada specification *Tasking_Properties* containing information on all tasks and protected objects in the system
- Create an Ada main program
- Create Ada tasks and protected objects based on the real-time properties of provided interfaces on leaf components
- Implement the provided interfaces on composite components based on delegate connectors to provided interfaces on constituent components
- Implement the Ada specifications of required interfaces based on the originating connectors.

Like the Functional View, the Concurrency View is an Ada package hierarchy with the same structure as the UML model. However, an artificial top level package called *CV* has been added to the Concurrency View package hierarchy to enable the Ada compiler to distinguish between Concurrency View Ada packages and Functional View Ada packages.

For each leaf component in the UML model, the vertical transformation will create an Ada package in the Concurrency View Ada package hierarchy. The specification of the Ada package will in most cases be

[2] In addition to the required interfaces, the implementation can of course make use of Ada packages defining (abstract) types for the parameters used in the signatures for the provided and required interfaces.

[3] The XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information via XML.

identical to the corresponding specification in the Functional View [4]. However, the implementation given in the Ada body will take into account the stereotypes and attributes of the provided interfaces. Basically, the implementation adds a layer [5] ensuring correct invocation of the interfaces according to their real-time properties. Upon correct invocation of an interface, the sequential implementation of the operation in the Functional View will be executed. As an example consider the generated Ada packages for the *TC_Router* component.

Ada specification for the *TC_Router*:

```
with Data_View;
package CV.Micro_OBOSS.Router.TC_Router is
  procedure Receive_TC(TC : in Data_View.TC_T);
end CV.Micro_OBOSS.Router.TC_Router;
```

Ada body for the *TC_Router*:

```
with Micro_OBOSS.Router.TC_Router;
with Tasking_Properties;
with Sporadic_Task;
package body CV.Micro_OBOSS.Router.TC_Router is
  package Spo_Task is new Sporadic_Task
    (Deadline   =>
       Tasking_Properties.
       CV_Micro_OBOSS_Router_TC_
                        Router_Receive_TC.Deadline,
     ...
     Parameter_T =>
       Data_View.TC_T,
     Operation   =>
       Standard.Micro_OBOSS.Router.
                        TC_Router.Receive_TC);
  procedure Receive_TC(TC : in Data_View.TC_T) is
  begin
    if not Spo_Task.Request_Sporadic_Operation(TC)
      then
      - - protected queue full
      ...
    end if;
  end Receive_TC;
end CV.Micro_OBOSS.Router.TC_Router;
```

The subprogram specification of the provided interface *Receive_TC* in the Concurrency View Ada specification is identical to the one in the Functional View Ada specification.

In the Concurrency View Ada body, *Receive_TC* inserts a request in a protected queue by calling *Spo_Task.Request_Sporadic_Operation*. The Ada package *Spo_Task* is an instantiation of the generic package *Sporadic_Task*, which creates a protected request queue and a sporadic task handling requests added to the queue. Arguments to the instantiation are real-time properties for *Receive_TC* such as deadline, MIAT, size of protected queue object, priority of sporadic task, and priority of protected queue object. These properties are located in the *Tasking_Properties* Ada package. In addition, the sporadic operation *Micro_OBOSS.Router.TC_Router.Receive_TC* defined in the Functional View as well as its parameter type *Data_View.TC_T* are used to instantiate the *Sporadic_Task.* This sequential implementation of *Receive_TC* will by the task be used to handle requests received from the protected queue.

In summary, the Concurrency View adds a layer consisting of a task and a protected queue object. Within this layer, the Functional View implementation of the *Receive_TC* interface is executed. The layer implements the sporadic properties of the operation by separating the request for the execution of the operation (performed by the calling thread by inserting the request in a protected queue) from the actual execution of the operation (performed by a different thread when the request has been extracted from the queue).

The Ada body for the generic package *Sporadic_Task* is listed below:

```
with Protected_Queue;
package body Sporadic_Task is
  package Request_Queue is new
Protected_Queue(...);
  function Request_Sporadic_Operation(Parameter : in
Parameter_T)
    return Boolean is
  begin
    return Request_Queue.Deposit(Parameter);
  end Request_Sporadic_Operation;
  task body Sporadic_Thread is
    ...
  begin
    loop
      - - Wait for a request to be available
      Request := Request_Queue.Extract;
      - - If the request was inserted during a burst,
      - - i.e. the queue was not empty
      if Request.Burst then
        - - The start time is the end time for
        - - the previous request
        Start_Time := End_Time;
      else
        - - The start time is the time at which the request
        - - was inserted into the queue
        Start_Time := Request.Time;
      end if;
      - - The end time is MIAT time units after start time
      End_Time := Start_Time + MIAT;
      - - Perform sporadic operation
      Operation(Request.Parameter);
      - - Check if the deadline has been missed
      Response_Time := Real_Time.Clock –
                          Start_Time;
```

[4] Operations with stereotype ‹‹cyclic›› or ‹‹blocking sporadic›› have no signatures in the Concurrency View

[5] This layer is called *Container* in the SAVOIR FAIRE Reference Architecture

```
        if Response_Time > Deadline then
          ...
        end if;
        - - Guarantee minimal inter-arrival time
        delay until End_Time;
      end loop;
    exception
      when The_Exception: others =>
        ...
    end Sporadic_Thread;
  end Sporadic_Task;
```

For each composite component in the UML model, the vertical transformation will:

- Create a Concurrency View Ada package (specification and body) for the composite component. In this package, the provided interfaces will be implemented by means of renaming declarations based on the delegate connectors from the provided interfaces of the composite component to provided interfaces of the constituent components. As an example, consider the generated Ada body for Router:

```
with CV.Micro_OBOSS.Router.TC_Router;
with CV.Micro_OBOSS.Router.TM_Router;
package body CV.Micro_OBOSS.Router is
  procedure Receive_TC(TC : in Data_View.TC_T)
     renames CV.Micro_OBOSS.Router.
                              TC_Router.Receive_TC;
  procedure Receive_TM(TM : in Data_View.TM_T)
     renames CV.Micro_OBOSS.Router.
                              TM_Router.Receive_TM;
end CV.Micro_OBOSS.Router;
```

- For each constituent component, create an Ada body implementing the (Functional View) Ada specification for the required interfaces. The implementation of a required interface depends on the type of the originating connector:
  - For a delegate connector, the implementation will be a renaming declaration based on the connection to a required interface on the composite component. As an example, consider the generated Ada body for the Ada specification for the required interfaces on *TC_Router*:

```
with Micro_OBOSS.Router_Required;
package body Micro_OBOSS.Router.
                              TC_Router_Required is
  procedure Handle_Display_TC(
                              TC : in Data_View.TC_T)
     renames Micro_OBOSS.Router_Required.
                              Handle_Display_TC;
end Micro_OBOSS.Router.TC_Router_Required;
```

  - For a dependency connector, the implementation will be a renaming declaration based on the connection to a provided interface on a sibling constituent component. As an example, consider the generated Ada body for the Ada specification for the required interfaces on Router:

```
with CV.Micro_OBOSS.Spacelink;
with CV.Micro_OBOSS.Display_Manager;
package body Micro_OBOSS.Router_Required is
  procedure Forward_TM(TM : in Data_View.TM_T)
     renames
       CV.Micro_OBOSS.Spacelink.Receive_TM;
  procedure Handle_Display_TC(
                              TC : in Data_View.TC_T)
     renames CV.Micro_OBOSS.Display_
                              Manager.Handle_TC;
end Micro_OBOSS.Router_Required;
```

As can be seen from the examples, connectors are implemented as renaming declarations. It is worthwhile noticing, that the implementation of a dependency connector is always a renaming from a Functional View specification of a required interface to a subprogram for a provided interface in the Concurrency View. Thus, a subprogram in the Functional View can call subprograms in the Concurrency View (as a result of dependency connectors in the UML design), but *never* call a subprogram in the Functional View.

Although the transformation to the Concurrency View has been based on an example of a sporadic interface, the idea is hopefully clear: The Concurrency View provides the layer implementing the real-time properties of provided interfaces, while the Functional View provides the sequential implementation of the interfaces.

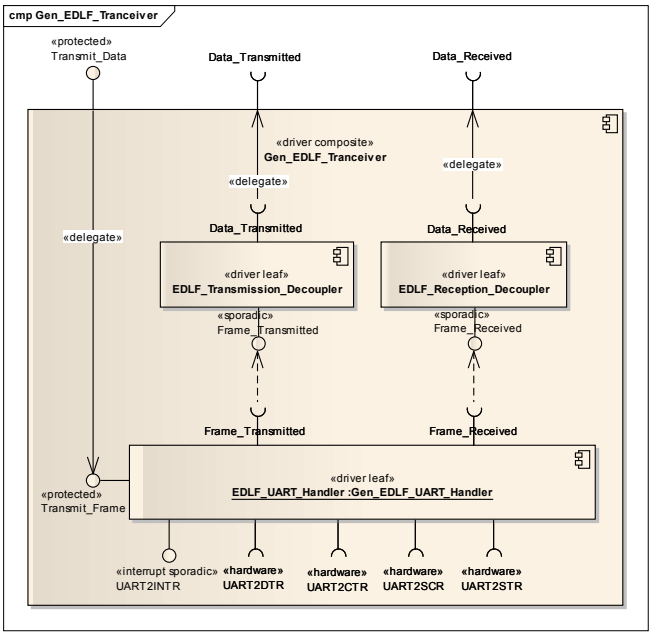


**Figure 2 UML model of Gen_EDLF_Tranceiver**

## 3 Extension of ASSERT

Compared to ASSERT and TASTE, the Terma modelling tool chain has been augmented with a number of features facilitating the modelling of components interacting with hardware, while still ensuring that the resulting Concurrency View is Ravenscar compliant.

The most important extension is the stereotype ‹‹interrupt sporadic›› that allows the designer to model interrupt service routines. Figure 2 shows a common component *Gen_EDLF_Tranceiver* shared between the two ASIM instrument designs. This component provides high level interfaces for sending and receiving so-called EDLF frames (telemetry frames), while leaving the low level byte handling to the child component *EDLF_UART_Handler*, whose ‹‹interrupt sporadic›› interface *UART2INTR* handles interrupts for reception and transmission of bytes via a UART. It would not be possible to model this behaviour in ASSERT.

The Terma modelling tool chain has also been extended to support a Watchdog protocol for securing aliveness of the tasks in the system.

All cyclic tasks will call a subprogram *Heartbeat_Response* as part of their invocation. All Ada packages containing a sporadic task will provide a subprogram *Request_Heartbeat*, which will request the embedded task to call the subprogram *Heartbeat_Response*. The subprogram *Heartbeat_Response* is a formal parameter to the generic packages creating cyclic and sporadic tasks (see description of Concurrency View above) and the designer is free to choose the actual subprogram to be called. The Concurrency View also provides a subprogram *Distribute_Heartbeat_Requests*, which will call the *Request_Heartbeat* subprogram in all Ada packages containing a sporadic task.

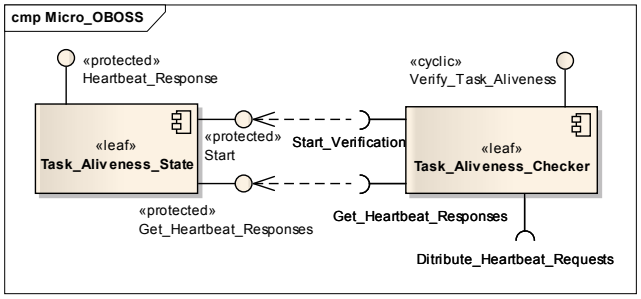


**Figure 3 Example of Watchdog protocol**

Figure 3 shows an example of how the Watchdog support can be utilized in a design. With a period large enough to allow all tasks to respond, the cyclic operation *Verify_Task_Aliveness* will: End the current verification session by calling *Get_Heartbeat_Responses* and checking if all tasks in the system have responded, and then start a new verification session by calling *Start* and *Distribute_Heartbeat_Requests*. The component *Task_Aliveness_State* keeps track of tasks that have responded. When *Start* is called, the set of responders is set to empty, and each time a task calls *Heartbeat_Response*, the task is added to the set of responders. The operation *Get_Heartbeat_Responses* simply returns the set of responding tasks.

## 4 Current Status of the Development

At the time of writing (May 2012), a design of the software for each instrument has been constructed. These two designs have not yet any Functional Views. Still, the constructed component models have been used for:

- Requirements tracing: Software requirements have been imported into the UML models from DOORS, and the designers have traced components to requirements by setting up links. This trace information has been automatically extracted from the component models as trace matrices, which have been checked for completeness and included in the design documentation.
- Rule compliance validation: A number of rules have been set up for the constructed designs. For instance, all provided and required interfaces on leaf components must have a description. A validation tool has been developed that validates these rules.
- Completeness checking: Completeness properties of the designs are automatically validated by the validation tool, e.g. that all required interfaces are connected to provided interfaces.
- Consistency checking: The consistency of the designs is automatically checked by the validation tool, e.g. that a provided interface is not connected to another provided interface.
- Schedulability analysis: Based on the Interface Views of the designs, which include specified real-time attributes and estimated worst-case execution times, schedulability analyses have been performed showing that all Ada tasks in the instrument software will meet their deadlines.

The implementation of the vertical transformation from the component model (containing both the Interface View and the Functional View) to the Concurrency View is almost finished.

It is considered a key issue that the generated Ada packages are easy to read and understand. This is necessary because the automatically generated code will be tested as if it was manually coded. In fact, unit testing, integration testing and system testing will be performed as though all source files were manually coded.

## 5 Conclusion

The Terma modelling tool chain has been developed gradually as part of the design of the ASIM instrument software. The most powerful part, namely the transformation of the Interface View and Functional View into the Concurrency View has not yet been utilized, because the Functional Views for the ASIM instrument designs have not yet been completed. The transformation has been tested on a number of simple models (like the Micro OBOSS model) and the tests show that the generated

code will compile, link, and run successfully. However, it should be noted that the tool set represents work in progress.

So far, the designs have benefitted from tool support for generation of trace matrices, rule compliance validation, completeness checking, consistency checking, and schedulability analysis. This has undoubtedly led to a higher quality of the designs compared to what would have been possible without tool support.

The effort that has gone into the development of the tool set has not been measured separately, but has been accounted as design work. To some extent, the tools development has been integration of existing pieces. For the vertical transformation, the generic task packages that are used for the generation of cyclic and sporadic tasks were taken from the OBOSS framework. Terma has a schedulability analysis tool developed on another project. This tool was adapted to be able to take input extracted from a design.

In summary, the Terma modelling tool chain has improved the quality of the ASIM instrument designs at a reasonable low development cost. Furthermore, the tool chain will reduce the time for the source code production, because the concurrency layer consisting of Ada tasks and protected objects is automatically generated.

## References

[1] SAVOIR-FAIRE On-board software reference architecture, issue 1.0, 10. June 2010

[2] ASSERT Project, http://www.assert-project.net/

[3] TASTE, http://www.assert-project.net/-TASTE-

[4] A. Burns & A. Wellings (2001), *Real-Time Systems and Programming Languages*, Addison-Wesley

# The Benefits of Using SPARK for High-Assurance Software

***Trevor J. Jennings***
*Altran Praxis Ltd., 20 Manvers Street, Bath, UK, BA1 1 PX.; Tel: +44 1225 466991;*
*email: Trevor.Jennings@altran-praxis.com*

## Abstract

*SPARK is a contract-based subset of Ada which is unambiguous and suitable for rigorous static analysis. It has been extensively used in industrial applications where safety and security are paramount. We will consider the benefits of using SPARK and how its features and supporting tools aid in the development and delivery of high-assurance software.*

*SPARK supports formal verification of program properties, for instance the proof of the absence of run-time exceptions or the proof that a program satisfies a security policy. Credit can be taken for mathematical proof as an alternative to testing in DO-178C and we consider how the SPARK tools can be used in this context.*

*Keywords: Ada, SPARK, Static Analysis, Formal Verification, Program Proof, DO-178C.*

## 1 Introduction

This article is a summary of a tutorial to be given at Ada Europe 2012.

High-assurance software typically has the following characteristics:

- Zero tolerance of defects in-the-field
- Potential for catastrophic loss
- Presence of a regulator and/or legal liability
- Need to generate evidence of fitness-for-purpose before first deployment
- "Patch it later" is not possible!

This is totally different from systems which can evolve ultra-reliability over many years and upgrades.

In the past, we have managed to tolerate defective software in safety-critical systems because, to some extent, software is part of a larger engineering system that is inherently redundant and fault-tolerant, e.g. a commercial aircraft – 4 engines, 2 wings, 3 hydraulic systems, stable glide behaviour, etc.

Hitherto, the environment has not been considered malicious but the "playing field" is changing. In the world of secure systems, a different set of assumptions apply:

- We are programming "Satan's computer" not "Murphy's computer" [1]
- Systems are increasingly networked and connected, leading to a sharp rise in complexity
- The system will be attacked by an arbitrarily intelligent and well-resourced opponent.

Defective software does not stand up well in this environment. Consider, for example, the commonly-occurring "buffer overflow" problem.

There are systems written in many different programming languages which are highly reliable, for instance the Linux Kernel, Apache, and LaTeX. Such systems tend to evolve ultra-reliability over many years and releases but this approach does not scale very well and is not good enough if we want high-assurance at first deployment.

Just testing a program extensively is not a complete answer. For a start it is not generally possible for most programs as their state space is vast; testing only ever touches a tiny fraction of the paths and input values. To claim a statistical reliability of N, how much testing to you need to do? For example, commercial aircraft aim for 1 failure in $10^9$ flying hours: how much testing is required for this level of reliability?

## 2 Two Independent Views

"Those who want really reliable software will find that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper", [2] Dijkstra in 1972.

"Some people argue that the easy defects are found by inspections and the difficult ones are left for testing, but I have seen no data to support this. The PSP data show that, for every defect type and for every language measured, defect-repair costs are highest in testing and during customer use. Anyone who seeks to reduce development cost or time must focus on preventing or removing every possible defect before they start testing.", [3] Humphrey in 2005.

It is interesting to note that nothing much changed in the 33 years that elapsed between these two quotes.

But how can you prevent and detect defects before testing or "observation-based" (dynamic) verification? By using static methods – analysis of design artefacts prior to

deployment or testing. This is identical to everyday practice in all mature engineering disciplines.

Data from the SEI indicates that project cost-overrun and schedule-overrun both correlate strongly with pre-test defect rate. This should be a massive incentive to reduce defects early, regardless of assurance level.

At high-assurance levels the situation is more pronounced, since "late" defects are more expensive and time-consuming to fix.

## 3 SPARK and Static Analysis

Static analysis, also known as static verification, is the verification of system properties based on analysis of design artefacts (e.g. source code), without observation or "testing" of the running system. It helps prevent mistakes and discovers mistakes sooner rather than later (i.e., during testing). Static analysis and verification cover a broad range of activities; it may be done manually or be tool supported and to varying depths of analysis. Examples in increasing level of depth are:

- Enforcing your coding standard
- Fagan inspections
- Code review
- Automated detection of runtime errors
- Proof of code properties; for instance, compliance with a security policy

Ideally, static verification should deliver analyses which are:

- Deep (it tells you something useful)
- Sound (with no false-negatives)
- Fast (it tells you now)
- Complete (with as few false-positives as possible)
- Modular and Constructive (it works on incomplete programs so that it can be applied early)

A well defined static analysis will find all cases where a program does not satisfy the criteria being checked by the analysis, on all paths, for all possible input values. It is an analysis rather than testing for specific values.

However there is a catch, the ability to deliver accurate, deep and sound static verification critically depends on the language that is being analysed. Most languages were not designed with static verification as a primary design goal. And it shows! With contemporary unsubsetted languages, you just can't deliver all of the above 5 goals.

Static analysis is used to answer questions of the sort:

"What does this program mean?" Or, more specifically, "Does my program have property X?" (e.g. "no buffer overflows"). There should be only one answer; a tool that is unable to determine the answer is of limited use and one that silently gives you the wrong answer cannot be trusted.

Ambiguity in language definition hinders the ability to reason, or just leads to unsoundness. The standard definitions of all common unsubsetted programming languages are ambiguous, e.g. unspecified and undefined behaviours in C and implementation-dependent and implementation-defined behaviours in Ada. The Standards are important, because that's what the compilers implement.

Ambiguity is a terrible curse from the point of view of a verification tool, since it impacts soundness and completeness.

To gain market share, most tools have to analyse the "whole language" or (worse) a dialect of a language; e.g. ISO 1990 C, or C "as implemented" by compilers X, Y and Z.

But everyone uses a subset! Has your project got a coding standard? Does it say "you must use every language feature"?

SPARK is a subset of Ada annotated with contracts. It is a programming language designed to deliver static verification that really is deep, sound, and as complete as possible, fast, constructive, and modular. It is a programming language with an unambiguous semantics.

To obtain the maximum benefits of using static analysis, SPARK suggests and encourages the use of a design philosophy for high-integrity software.

SPARK was specifically designed for use in embedded and real-time systems with typical characteristics:

- Hard real-time requirements
- Little or no Operating System on target (no disk or virtual memory)
- Fixed, known amount of storage

There are many industrial systems with these characteristics written in SPARK, but the SPARK language is increasingly being used to write software for much larger and more dynamic systems working on top of full operating systems.

The SPARK language also supports an annotated subset of the Ravenscar profile, RavenSPARK, for concurrent applications.

## 4 Abstraction and Refinement

The appropriate use of abstraction is the key to managing complexity and the SPARK contract language adds abstractions that are useful in static analysis and program proof in particular. These abstractions are over and above the computational abstractions provided by Ada.

The two main abstractions which are provided by SPARK contracts are own variables, and proof functions.

Additionally a global annotation completes the abstraction of an Ada subprogram specification so that it can be considered in isolation with the closure of all variables that it reads and may update. The specification of a subprogram may be further supplemented by formal pre and post conditions. The post condition is effectively a mathematical

abstraction of the effect (or of at least certain properties) of the subprogram.

Own variables are an abstraction of the internal state of a package and its private children. A package may have more than one own variable and the actual package variables are a refinement of an own variable.

The own variable abstraction provides a single name to represent possibly very many actual variables. In static analysis, particularly with program proof, users of the package declaring the own variable only refer to the own variable rather than the detail within.

Own variables are also used as an abstraction for external inputs and outputs, modelling them as mathematical sequences. Without this model inputs and outputs (and volatile variables in general) are difficult to reason about in static analysis and in formal proofs.

Proof functions are logical functions which can only be called within SPARK contracts. They allow properties of own variables to be specified and facilitate the abstraction of complex pre and post conditions, essential if significant proof of properties is to be completed and be understandable. Proof functions may also have an abstract and refined view.

# 5 Program Proofs – Formal Verification

For the highest levels of assurance formal mathematical proofs of a program's properties may be performed on SPARK programs. This is possible because the SPARK language is unambiguous.

The sorts of properties that might be proven range from the proof of absence of run-time exceptions, through safety or security properties to proof of functional correctness.

To prove a property of a subprogram a formal specification of the property is required which is placed within a post condition contract. The SPARK tools can then analyse the code of the subprogram and generate a set of theorems (Verification Conditions, or VCs). These must be proven to demonstrate that the subprogram meets is specification for all input values that meet its precondition contract. The SPARK toolset includes an automatic theorem prover, the Simplifier, to assist in discharging these proofs.

In DO-178C [4] credit can be taken for formal proofs. This can minimise the amount of testing and certainly reduce the amount of re-testing required as the subprogram has been proven to meet its specification.

# 6 Proof of Absence of Run-Time Exceptions

In SPARK, the proofs of type safety and the absence of run-time exceptions are particularly straightforward and their proof gives great benefits and increased reliability in a high-assurance system.

The Ada Language Reference Manual [5] specifies where a type safety or other run-time check must be applied. These check points define a formal specification which has been incorporated in to the SPARK Tools. The VCs to prove type safety and the absence of run-time exceptions are generated automatically without the need to add post condition contracts to describe these properties.

The proof of type safety and absence of run-time exceptions generates a lot of VCs but generally they are small and amenable to automatic proof. In industrial applications, our experience with well written code is that the Simplifier will prove over 95% of the type safety and absence of run-time exception VCs automatically.

Generally, the Simplifier discharges the proofs quickly and so we would recommend always performing these proofs, even before code review.

# 7 Safety and Security Critical Software

The SPARK language and Toolset optionally support information flow analysis. Information flow analysis determines or checks the flow of information between variables and can be particularly useful when analysing a secure system where a classified input should not flow to an unclassified output.

The information flow analysis can be enhanced by annotating a criticality or confidentiality level to important variables.

However, using post condition contracts one can do much more and prove that the software satisfies a security policy model.

## 7.1 Proof of a Security Policy

The Tokeneer Project, is an NSA-funded, highly secure biometric software system implemented in SPARK to conform to the Common Criteria EAL5 requirements.

The software implementation was proven to meet the specified security policy using the SPARK Toolset.

The results of the Tokeneer project have been made available by NSA to the software development and security communities to demonstrate that it is possible to develop secure systems rigorously in a cost effective manner.

More details can be found and the project artefacts may be downloaded from:
http://www.adacore.com/sparkpro/tokeneer.

# References

[1] R. Anderson and R. Needham, *Programming Satan's Computer*, Cambridge University Computer Laboratory

[2] E. Dijkstra (1972), *Turing Award Lecture*.

[3] Watts Humphrey (2005), *PSP – A Self-Improvement Process for Software Engineers*, Addison Wesley, page 141.

[4] DO-333 *Formal Methods supplement to* DO-178C, *Software Considerations in Airborne Systems and Equipment Certification* (2012), RTCA.

International Standards Organisation, *Ada 95 Reference Manual*, ANSI/ISO/IEC-8652:1995

# The Use of Proof and Generics in SPARK

***Trevor J. Jennings***
*Altran Praxis Ltd., 20 Manvers Street, Bath, UK, BA1 1 PX.; Tel: +44 1225 466991;*
*email: Trevor.Jennings@altran-praxis.com*

## Abstract

*SPARK is a contract-based subset of Ada which is unambiguous and suitable for rigorous static analysis. It has been extensively used in industrial applications where safety and security are paramount.*

*The SPARK language and Toolset support the formal verification of programs - a mathematical proof that a program satisfies its specification. We consider simple examples of the proof of properties in SPARK programs. In particular we will look at the proof of absence of run-time exceptions. We describe the use of SPARK contracts to specify the desired properties of a subprogram and the proof tools used to prove the resulting theorems, including the new counter-example finding tool, Riposte.*

*The SPARK language and tools now support generics which can be used to provide reusable components. A SPARK generic unit only needs to be proved once but may be used (instantiated) many times without requiring further proof. SPARK generics support a strategy of prove once and use many times.*

*Keywords: Ada, SPARK, Static Analysis, Formal Verification, Program Proof, DO-178C, Generics, Counter-examples.*

## 1 Introduction

This article is a summary of a tutorial to be given at Ada Europe 2012.

SPARK is a subset of Ada annotated with contracts. The SPARK language is unambiguous and therefore code written in SPARK may be verified, using a mathematical proof, against a formal specification. A specification may be written in the contract language of SPARK and the SPARK toolset is then able to generate theorems. These theorems take the form of verification conditions (VCs) which have to be proven to show that the implemented code satisfies the specification. The SPARK tools include an automatic theorem prover to assist in discharging the VCs.

In DO-178C [2] credit can be taken for formal verification and so, increasingly, in the future we expect much more use of proof techniques.

Formal verification may minimise the amount of testing and certainly reduce the amount of re-testing required as the subprogram has been proven to meet its specification.

## 2 Proof of Absence of Run-Time Exceptions

The proof of absence of run-time exceptions is particularly straightforward and gives great benefits in terms of increased reliability in a high-assurance system.

The Ada Language Reference Manual [1] specifies where a run-time check must be applied. These check points define a formal specification which has been incorporated into the SPARK Tools. The VCs to prove the absence of run-time exceptions are generated automatically without the need to add post condition contracts.

The proof of the absence of run-time exceptions generates many VCs but generally they are small and amenable to automatic proof. In industrial applications, our experience, with well written code, is that over 95% of the VCs are proven automatically.

Generally, the proofs are discharged quickly and so we would recommend always performing these proofs, even before code review.

## 3 Proofs of other Properties

To prove that a subprogram satisfies any property other than absence of run-time exceptions requires the property to be specified in a post condition contract. For example, the specification of the functional properties of an integer square root operation:

```
procedure I_Sqrt_P (N : in Integer; Root : out integer);
--# pre N >= 0;
--# post Root * Root <= N and
--#        (Root + 1) * (Root + 1) > N;
```

Here the lines beginning with --# form the SPARK contract (or annotation), and appear as a comment to an Ada compiler. The specification is given by the **post** contract. The SPARK tools will analyse the body of I_Sqrt_P and generate VCs; if proven, these demonstrate that the implementation satisfies the specification given in the **post** contract for all input values which satisfy the **pre** contract. Wherever I_Sqrt_P is called the SPARK tools will generate a VC which has to be proven to show that the actual parameter satisfies the **pre** contract placed on the formal parameter.

We can avoid the need for a pre contract by using the Ada type system. The operation can also be more naturally expressed as a function:

```
function I_Sqrt (X : Natural) return Natural;
--# return R => R * R <= X and
--#                (R + 1) * (R + 1) > X;
```

In a function a **return** contract is used rather than a **post** contract but it serves the same purpose. If I_Sqrt is called with an argument not constrained to type Natural, e.g, an Integer parameter, the SPARK tools will generate a VC to prove that the value of the actual argument is in the range of Natural.

# 4 Abstraction and Refinement

A specification such as the one in the last section is fine if we have full visibility of the types of the parameters and results but suppose we are dealing with an abstract data type or a package containing state. To deal with these situations SPARK has proof functions. Proof functions can only be called within SPARK contracts.

## 4.1 Type Abstraction

Consider the following abstract type declaration:

```
package P is
type T is private;

--# function Is_Sorted (V : T) return Boolean;

 function Is_Member (V : T; N : Natural) return Boolean;
 --# pre Is_Sorted (V);
 procedure Init (V : out T);
 --# post Is_Sorted (V);

 procedure Add (V : in out T; N : in Natural);
 --# pre Is_Sorted (V);
 --# post Is_Sorted (V) and Is_Member  (V, N);
private
  ... -- the private part
end P;
```

When an object is referenced in a pre-condition it refers to the initial value of the object; when it appears in a post-condition it refers to the final value of the object. It is possible to refer to the initial value of an object in a post-condition by appending a tilde to its name, e.g, V~.

In this example the Is_Sorted is a proof function which describes a logical state of an object of type T. An Ada function could be used rather than a proof function but it might not ever be called in the actual Ada code and it may be difficult to implement. The proof function Is_Sorted may only be called within a SPARK contract, whereas the Ada function Is_Member may be used in both SPARK contracts and Ada code.

Proof functions may also be used to provide abstractions of complex pre and post-conditions.

## 4.2 Data Abstraction

The same example may be cast as a data package.

```
  package Q
  --# own Abstract_State : A_Set;
  is
    --# type A_Set is abstract;
    --# function Is_Sorted (S : A_Set) return Boolean;

    function Is_Member (N : Natural) return Boolean;
    --# global in Abstract_State;
    --# pre Is_Sorted (Abstract_State);

    procedure Init;
    --# global out Abstract_State;
    --# post Is_Sorted (Abstract_State);

    procedure Add (N : Natural);
    --# global in out Abstract_State;
    --# pre Is_Sorted (Abstract_State);
    --# post Is_Sorted (Abstract_State) and
    --#      Is_Member (Abstract_State, N);
  end Q;
```

The own variable Abstract_State is an abstraction of the internal state of the package. The abstraction is a simple variable but the internal state of the package could be complex. The own variable may only be used within SPARK contracts.

To declare proof functions to represent logical states of the own variable requires that the own variable is given a type. Here it has been given a SPARK abstract type declaration because we do not want to reveal the internal structure of the data in the package.

The abstract own variable has to be refined into its constituent parts in the body of the package and the function and procedure specifications refined accordingly. The SPARK language provides the means to do these refinements and prove correspondence between the abstract and refined versions.

# 5 The SPARK Proof Tools

The SPARK Toolset contains a number of proof related tools that work in conjunction:

- The Examiner analyses the program text and generates the VCs and some axiomatic rules
- The Simplifier is an automatic theorem prover which takes the VCs generated by the Examiner and attempts to prove them but if it is unable to prove a VC it leaves it in a simplified form
- SPARKBridge – built around the Victor tool – converts the VCs generated by the Examiner into SMT-Lib format which is used by many proof tools
- Alt-Ergo is an SMT solver which uses different techniques to the Simplifier in attempting an automatic proof
- SPARKSimp is a tool which assists in parallelising the proving of VCs. It determines which VCs are to be proven and spawns multiple instances of the Simplifier to prove them. If a VC is not proven by the Simplifier it can spawn Victor and henceforth Alt-Ergo to attempt the proof

- Riposte is a counter-example generator. If a VC is not proven it may be because it is false. Given such a VC, Riposte will find a counter- example – that is, a set of values for which the VC is false
- The Proof Checker is an interactive theorem prover which can be used to dispatch particularly tricky proofs that need human insight
- POGS, a proof obligation summariser, collects the results from VC generation and the outputs from the provers to summarise which VCs have been proven and by which tool. It also highlights any false VCs.

As well as these tools shipped with the SPARK Toolset there is an interface to the Isabelle generic proof assistant [3], that takes the VCs generated by the Examiner and translates them to the Isabelle format, HOL-SPARK [4]. The translator is part of the Isabelle package which can be downloaded from

http://www.cl.cam.ac.uk/research/hvg/isabelle/.

The documentation for HOL-SPARK is available form

http://isabelle.in.tum.de/library/HOL/HOL-Word/HOL-SPARK/Manual/document.pdf

# 6 SPARK Generics

A new feature of the SPARK language is an annotated subset of Ada generics which will facilitate the production of reusable verified components. In particular it is planned to provide a SPARK container library in the future.

SPARK generics excludes the features of Ada that are not allowed in SPARK but still leaves a powerful subset. We have ensured that the features required to support a container library have been included in this subset. The principal of prove once and use many times is fundamental to the definition of SPARK generics.

## 6.1 Main Features

The main features of SPARK generics are:

- Library–level generic subprograms and packages may be declared
- A generic declaration may also appear in the visible part of a library-level generic package declaration
- A generic subprogram instantiation may appear anywhere a subprogram can legally be declared in SPARK and similarly for a generic package instantiation
- Additionally a generic package instantiation may appear in the private part of a package (this is already a legal place for a subprogram declaration in SPARK)
- A generic declaration may have an instantiation check
- A generic subprogram declaration and its instantiation may have pre and post/return annotations

## 6.2 Formal Generic Parameters

The formal generic parameters supported by SPARK are:

- Private and limited private types
- Discrete types (excluding Boolean)
- Signed integer types
- Modular types
- Floating and fixed point types
- Constrained and unconstrained array types
- Object parameters are supported but they must be of mode in and instantiated with a SPARK constant expression
- Subprogram parameters are supported but they cannot refer to global variables

## 6.3 Proofs for Generics

A generic declaration may require proofs arising from assumptions made in the declarative part; for instance:

```
generic
  type T is range <>;
  --# check T'First <=0 and T'Last >= 10;
package G is
    subtype Small_T is range 0 .. 10;
…
end G;
```

Here VCs have to be generated to show the 0 and 10 are in the range of T. To prove the VCs the instantiation check is required.

An instantiation check is declared on the generic but checked at the point of instantiation by generating appropriate VCs.

The body of a generic unit has to be proved in terms of the properties of its generic parameters and the assumption that the instantiation check of its declaration is True, but otherwise it is no different to the proof of a non-generic body.

Properties proven for a body of a generic hold for all instantiations of a generic provided the instantiation check is satisfied.

If an instantiation of a subprogram does not have its own pre or post conditions then the corresponding pre or post condition from the generic declaration applies.

If the instantiation of a subprogram has its own pre or post conditions, then VCs are generated as appropriate to prove:

pre_instantiation => pre_generic

pre_instantiation and post_generic => post_instantiation.

If the generic declaration does not have a pre-condition it is assumed to be True. If a generic procedure declaration

does not have a post condition contract then it is also considered to be True. If a generic function declaration does not have a return contract then it is assumed to be empty and the return contract from the instantiation is used directly but with a warning that it cannot be checked.

### 6.4 Generic Subprogram Declaration

An example of a generic subprogram declaration:

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

Note the instantiation check, which must be satisfied by every instantiation, and the return contract applied to the subprogram specification.

### 6.5 Generic Subprogram Instantiation

The above example declaration may be instantiated as:

```
type Actual_T1 is range 0 .. 10;
type Actual_T2 is range 0 .. Actual_T1'Last  * * 2;
function My_Square
--# pre X  > 1;
--# return R => R = T2 (X * X) and R >= 4;
is new Square (T1 => Actual_T1, T2 => Actual_T2);
```

Note that the instantiation check is satisfied, and by strengthening the pre-condition contract on the instantiation we are able to also apply a stronger return contract.

#### 6.6 Ada.Unchecked_Conversion

In SPARK the predefined function Ada.Unchecked_Conversion is considered to have the declaration:

```
generic
type Source is private;
type Target is private;
function Unchecked_Conversion (S : Source)
   return Target;
```

Note that the pre-condition contract is considered to be True and the return contract to be empty.

An instantiation of Ada.Unchecked_Conversion may have pre and return contracts applied:

```
type Byte is mod 256;
type Status is (S0, S1, S2, S3, S4);
subtype Running is Status range S1 .. S4;

function Byte_To_Status
--# pre S > 0 and S <= 4;
--# return R => R in Running;    -- Generates a warning
is new Unchecked_Conversion
  (Source => Byte, Target => Status);
```

Here we have added the pre-condition that S > 0 and S <= 4 and given this pre-condition we assert that the result is in Running. This will be assumed to be the case wherever Byte_To_Status is called and the pre-condition is satisfied.

## 7 Summary

Large industrial projects have been written in SPARK, and its proof tools have been successfully used to formally verify properties of the resulting programs.

The credits available for formal verification in DO-178C make the use of proof techniques increasingly attractive.

The addition of generics to SPARK facilitates the production of reusable formally verified components.

## References

[5] International Standards Organisation, *Ada 95 Reference Manual*, ANSI/ISO/IEC-8652:1995

[6] DO-333 *Formal Methods supplement to* DO-178C, *Software Considerations in Airborne Systems and Equipment Certification* (2012), RTCA.

[7] Tobias Nipkow, Lawrence C Paulson, Markus Wenzel (2011), *Isabelle HOL A Proof Assist for Higher-Order Logic*, Springer Verlag.

[8] Stefan Berghofer (2011), *The HOL-SPARK Program Verification Environment*, Secunet Security Networks AG.

# 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 #111: The Distributed Systems Annex, Part 5—Embedded Name Server

**Thomas Quinot, AdaCore**

**Abstract.** This is the fifth in a series of Gems introducing the facilities defined by the optional annex for Distributed Systems of the Ada Reference Manual (Annex E). In the previous instalment [published in AUJ Vol. 31, No 4, December 2010 —lmp], we showed how to integrate DSA code as a stand-alone Ada library in a C/C++ application.

In this installment, we show how the DSA name server can be embedded in the main partition, rather than started as a stand-alone process.

### Let's get started…

In the first installment in this series of DSA Gems, we used an application managing a public bulletin board as a good example of a client-server design. We used the following configuration:

```
configuration Dist_App is
  pragma Starter (None);
  -- User starts each partition manually

  ServerP : Partition := (Bulletin_Board);
  -- RCI package Bulletin_Board is on partition ServerP

  ClientP : Partition := ();
  -- Partition ClientP has no RCI packages

  for ClientP'Termination use Local_Termination;
  -- No global termination

  procedure Display_Messages is in ServerP;
  -- Main subprogram of master partition

  procedure Post_Message;
  for ClientP'Main use Post_Message;
  -- Main subprogram of slave partition
end Dist_App;
```

and we obtained two executables, one for each partition (serverp and clientp), by issuing the following command:

```
po_gnatdist dist_app
```

With the above po_gnatdist configuration, running the application requires three steps:

- Start po_cos_naming, the PolyORB name server. On startup, an object reference is displayed, which must be passed to all partitions in the environment variable POLYORB_DSA_ NAME_SERVICE or through a PolyORB configuration file.
- Start serverp, the server partition, which will register its RCI package (Bulletin_Board) with the name server.
- Start clientp, the client partition, which will query the name server for the location of the RCI.

Simplifying the process

To make this whole process easier, you can instead direct po_gnatdist to embed the name server within the main partition (serverp). This is achieved by saying:

```
pragma Name_Server (Embedded);
```

in the configuration file.

You can then start serverp without an external name server: it is now included within the partition. You still need to convey the location of the name server to clients. From within the server partition, this information can be retrieved by querying the PolyORB run-time parameters:

```
Put_Line ("ServerP started, embedded name server is at:");
Put_Line (PolyORB.Parameters.Get_Conf ("dsa",
                                       "name_service", ""));
```

This outputs a string of the form:

```
IOR:<...long series of hex digits>
```

which encodes all required information. You can then start the client partition by specifying this value in the POLYORB_DSA_NAME_SERVICE environment variable. In Bourne shell syntax, this translates to:

```
POLYORB_DSA_NAME_SERVICE=IOR:<...> ./clientp
```

Using the Windows cmd shell, this would be:

```
set POLYORB_DSA_NAME_SERVICE=IOR:<...>
client
```

Passing the name server information in a file

POLYORB_DSA_NAME_SERVICE can also indicate a file name, prefixed with the string "file:". So, if you modify serverp to output the name service reference to a file "ns.txt", you can start the client using

```
POLYORB_NAME_SERVICE=file:ns.txt ./clientp
```

Using a well-known port

*Note: this requires the latest development version of PolyORB.*

It is sometimes inconvenient to have to transport a string value or a file from the server to the client, and to have to update it each time the server is restarted. Using appropriate PolyORB run-time configuration directives, you can force the server to listen for network connections at a fixed location.

The following configuration forces the server to listen on port 8889:

```
[iiop]
polyorb.protocols.iiop.default_port=8889
```

Once this is set for the server, you can direct the client to that location by setting POLYORB_DSA_NAME_SERVICE to:

```
corbaloc:iiop:1.2@<hostname>:8889/_NameService
```

The included [1]start_server and start_client scripts provide a demonstration of this facility.

# Gem #107: Preventing Deallocation for Reference-counted Types

**Christoph Grein, Ada Magic**

**Abstract:** This Gem was contributed as a followup to an earlier Gem. Christoph explores ways in which the API for reference-counted types can be made safer by taking advantage of some Ada 2005 features.

### Let's get started…

In Gem #97 [published in AUJ Vol. 32, No 1, March 2011 —lmp], a reference-counting pointer was presented, where a Get function returns an access to the data. This could be dangerous, since the caller might want to free the data (which should remain under control of the reference type). In this Gem, we present a method to prevent the misuse of the result of Get.

Let's repeat the relevant declarations:

```
type Refcounted is abstract tagged private;
type Refcounted_Access is access Refcounted'Class;

type Ref is tagged private;  -- our smart pointer

procedure Set (Self: in out Ref; Data: Refcounted'Class);
function  Get (Self: Ref) return Refcounted_Access;

private

type Ref is new Ada.Finalization.Controlled with record
   Data: Refcounted_Access;
end record;
```

The function Get lets us retrieve and modify the accessed object. The problem with this function is that it compromises the safety of the pointer type Ref, in that a caller might copy the result access object and deallocate the accessed object:

```
Copy: Refcounted_Access := Get (P);
Free (Copy);
```

where Free is an appropriate instantiation of Unchecked_Deallocation.

To cure the situation, we no longer return a direct access to the data. Instead we define an accessor, a limited type with such an access as a discriminant, and let Get return an object of such a type:

```
type Accessor (Data: access Refcounted'Class) is
   limited null record;
function Get (Self: Ref) return Accessor;
```

Making the type limited prevents copying, and access discriminants are unchangeable. The discriminant also cannot be copied to a variable of type Refcounted_Access. The result is that the discriminant can be used only for reading and writing the object, but not for deallocation. Thus we have achieved our goal of making accesses safe.

A user might now declare some type derived from Refcounted and change the value of the accessed object like so:

```
declare
   type My_Refcount is new Refcounted with record
      I: Integer;
   end record;

   P: Ref;

begin
   Set (P, My_Refcount'(Refcounted with I => -10));
   My_Refcount (Get (P).Data.all).I := 42;
end;
```

This view conversion to My_Refcount will incur a tag check that will succeed in this example. In general, you have to know the type with which to view-convert in order to access the relevant components. An alternative is to declare a generic package like the following:

```
generic
   type T is private;
package Generic_Pointers is
   type Accessor (Data: access T) is limited private;
   type Smart_Pointer is private;
   procedure Set (Self: in out Smart_Pointer; Data: in T);
   function  Get (Self: Smart_Pointer) return Accessor;
private
   ... implementation not shown
end Generic_Pointers;
```

Instantiate with type Integer and the last line becomes instead:

```
Get (P).Data.all := 42;
```

So how do we implement the function Get? This is quite straightforward in Ada 2005, using a function returning a limited aggregate. (Note that in Ada 95, limited objects were returned by reference, whereas in Ada 2005 limited function results are built in place.)

```
function Get (Self: Ref) return Accessor is
begin
   return Accessor'(Data => Self.Data);
end Get;
```

Alas, we are not yet completely safe. To see this, we have to consider in detail the lifetime of the Accessor objects. In the example above, the lifetime of Get (P) ends with the statement and the accessor is finalized. That is, it ceases to exist (in Ada vernacular, the master of the object is the statement). So, tasking issues aside, nothing can happen to the accessed object (the integer in our example) as long as the accessor exists.

Now consider a variant of the above. Imagine we have a pointer P whose reference count is 1, and let's extend the accessor's lifetime:

[1] Included in the Gem page at http://www.adacore.com/adaanswers/gems/gem-111-the-distributed-systems-annex-part-5-embedded-name-server/

```
declare
   A: Accessor renames Get (P);
begin
   Set (P, ...);  - - allocate a new object
   My_Refcount (A.Data.all).I := 42;  -- ?
end;  - - A's lifetime ends here
```

In this example, the master of the accessor is the block (and there are other ways to make the lifetime as long as one wishes). Now in the block, the pointer P is given a new object to access. Since we said that P was the only pointer to the old object, it's finalized with disastrous effect: A.Data is now a dangling pointer granting access to a nonexistent object until the end of the declare block.

(Note that this issue also existed in the original GNATCOLL.Refcount implementation.)

To cure the situation, we have to prevent the deallocation. That suggests increasing the reference count with the construction of an accessor and decreasing the count when the accessor is finalized again. The easiest way to accomplish this is to piggyback upon the properties of the smart pointer type:

```
type Accessor (Data: access Refcounted'Class)
      is limited record
   Hold: Ref;
end record;

function Get (Self: Ref) return Accessor is
begin
   return Accessor'(Data => Self.Data, Hold => Self);
end Get;
```

Incidentally, as a final note, the type Accessor should probably be declared as limited private, to avoid the possibility of clients constructing aggregates (which, by the way, would be quite useless).

# Gem #123: Implicit Dereferencing in Ada 2012

## Christoph Grein, Ada Magic

**Abstract:** This Gem discusses the use of features added in Ada 2012 that simplify accessing and updating the elements of containers.

**Let's get started…**

In Gem #107, we presented an accessor for safely referencing objects stored in a container. The example concerned a reference-counted pointer, but such accessors can be defined on any kind of container.

An advantage of using accessors rather than simple access types is that the former cannot be used to deallocate the designated object. However, safety always comes with a cost, in this case in the form of awkward syntax. In this Gem, we show how some features of Ada 2012 can be used to simplify the syntax.

Consider a prototypical container as an example:

```
generic
   type Element is private;
   type Key     is private;
   with function Key_of (E: Element) return Key;
package Containers is

   type Container is private;

   type Accessor (Data: not null access Element)
      is limited private;

   procedure Put (C: in out Container; E: in Element);

   function  Get (C: Container; K: Key) return Accessor;

private
   ... implementation not shown
end Containers;
```

The container holds elements that can be retrieved via some kind of key. How elements are stored and retrieved is not of interest here. What is important is that Get grants direct access to the stored element (in other words, Get does not return a copy). This is crucial if you have a big object and only want to update a component:

```
Get (Cont, My_Key).Data.Component := Some_Value;
```

Note that if the discriminant were to be defined as

```
not null access constant Element
```

then the accessor would allow only read access. Also, the null exclusion guarantees that an accessor will always reference an object.

This syntax is quite verbose, but Ada 2012 provides a new feature that helps simplify it, namely the Implicit_Dereference aspect.

Side Note: Ada 2012 has added a general mechanism called an aspect specification that allows defining various characteristics of declarations, called aspects, as part of the declaration itself. For example, representation attributes can now be specified by using an aspect specification rather than a separate attribute definition.

Here is how the Implicit_Deference aspect would be specified for our Accessor type:

```
type Accessor (Data: not null access Element)
   is limited private
with Implicit_Dereference => Data;
```

A type defined with this aspect is called a reference type, and an object of such a type is a reference object. The use of this aspect allows us to reduce the statement to:

```
Get (Cont, My_Key).Component := Some_Value;
```

Note that the call Get(Cont, My_Key) is overloaded: its result can be interpreted as either an accessor value or the accessed object itself, and the compiler resolves this based on the context of the call. (This is the reason the Implicit_Dereference aspect cannot be used on the reference-counted pointer in Gem #107, where a type conversion is needed, because the argument of a type conversion must be resolved independently of the context.)

You might argue that this is not a significant simplification. However, this is not the end of the story. We're not interested so much in how to get an accessor value (the result of function Get) as we are in getting to the elements themselves. It happens that we can elide the call to Get by means of another

aspect, called Variable_Indexing, that's applied to the Container type:

```
type Container is tagged private
   with Variable_Indexing => Get;
```

The result type of the Variable_Indexing function must be a reference type. It's worth noting that the name given in an aspect specification may denote something declared later. In this case it's a forward reference to Get, which is declared after the type.

Also, the type to which the Variable_Indexing attribute is applied must be tagged. Being a tagged type, this allows the Object.Operation notation, leading to:

```
Cont.Get (My_Key).Component := Some_Value;
```

Given the Variable_Indexing aspect that specifies Get, this can now be further reduced to simply:

```
Cont (My_Key).Component := Some_Value;
```

Effectively what we get is direct access to container elements, as though the container were a kind of array indexed by the key. In fact, it's possible to use the indexed name alone in a context requiring a variable of the element type, such as an assignment statement:

```
Cont (My_Key) := Some_Element_Value;
```

So, by combining the new aspects Implicit_Dereference and Variable_Indexing we get a concise and much more readable syntax for manipulating container elements.

Incidentally, there's also a companion aspect to Variable_Indexing called Constant_Indexing, that can be used to grant read-only access to element values. In that case, the associated function is not required to return a reference type, because all functions return a constant result.

# 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*