ADA USER JOURNAL

Volume 33
Number 1
March 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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

In this first issue of volume 33 of the Ada User Journal, I would like to refer our readers to a change in the Journal's Editorial Team. Marco Panunzio, Journal's News Editor since December 2008, is leaving the Journal, as he prepares to embark in a new career. I am sure that you all join me in thanking Marco for his commitment and effort, and wishing him all the success in this new period of his professional life.

At the same time, I am pleased that Jacob Sparre Andersen, from Denmark, has accepted the challenge to move from the role of news contributor to the role of editor, volunteering some of his time to the preparation of the Journal; I express thanks and welcome Jacob to the job. In order to allow for a smooth transition, the News Digest in this issue was jointly produced by Marco and Jacob.

Continuing with contents of the Journal, the readers will also find the usual Calendar and Forthcoming Events sections, provided by Dirk Craeynest. The latter with the advance information of the 17th International Conference on Reliable Software Technologies – Ada-Europe 2012 that will take place next June in Stockholm, Sweden and the call for contributions to the 2012 SIGAda conference, under the name of High Integrity Language Technology. I would like to particularly point out the rich program of Ada-Europe 2012, of which I highlight the three keynote talks and the two panels, but also mention the 15 referred scientific papers and 9 industrial presentations, the rich set of tutorials, and the special Ada in Motion session. Reasons more than enough for me to exhort all Ada practitioners to attend this year's Ada-Europe conference

As for the technical contents of the issue, we continue the publication of the Ada 2012 Rationale, with the chapter on Expressions. This chapter describes and explains the new forms of expressions introduced in the language: if expressions, case expressions, quantified expressions and expression functions. One of the main reasons for the introduction of these new forms is to facilitate formulating contracts, but I am certain that these new forms will also prove to be valuable on their own and will augment the already rich set of Ada features.

The issue also provides an article by Bo Sandén, from the Colorado Technical University, USA, presenting the entity-life modeling design approach for multitasking software, and its application to Ada. As we all know, building multitasking software is a challenging chore (being a correct and careful design a cornerstone), thus it is important to have available sound and simple design approaches.

Finally, we conclude with two gems by Emmanuel Briot, on the implementation of the visitor pattern and overridable class attributes in Ada.

Our best wishes for (Ada) 2012,

Luís Miguel Pinho
Porto
March 2012
Email: lmp@isep.ipp.pt

Quarterly News Digest

Marco Panunzio

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

Jacob Sparre Andersen

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

Contents

Ada-related Organizations

Final draft version of the Ada 2012 standard

From: Thomas Løcke
Date: Thu, 1 Dec 2011
Subject: Final version of the Ada 2012 standard released
URL: http://ada-dk.org/2011/12/final-version-of-the-ada-2012-standard-released/

Draft 14 of the Ada 2012 standard is going to be the final version of what will end up as Ada 2012, so if you have any issues with the standard, now is the time to speak up.

[http://www.ada-auth.org/standards/ada12.html —mp]

If you want to know more about all the shiny new stuff in Ada 2012, then I'd suggest taking a look at the Ada 2012 Language Reference Manual and/or the Annotated Ada 2012 Language Reference Manual.

[http://www.ada-auth.org/standards/12rm/html/RM-TTL.html

http://www.ada-auth.org/standards/12aarm/html/AA-TTL.html —mp]

Thanks to Marc C. for bringing this to my attention.

Ada-related Events

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

FOSDEM 2012 — Presentations and some reactions

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Tue, 7 Feb 2012 22:07:00 +0000
Subject: FOSDEM 2012 - Presentations Ada Developer Room on-line
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

** All presentations available on-line **

Ada Developer Room at FOSDEM 2012

(Ada at the Free and Open-Source Software Developers' European Meeting)

Saturday 4 February 2012

Université Libre de Bruxelles (U.L.B.), Solbosch Campus, Room AW1.121

Avenue Franklin D. Roosevelt Laan 50, B-1050 Brussels, Belgium

Organized in cooperation with Ada-Europe

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/12/120204-fosdem.html

---

All presentations at the Ada Developer Room, held at FOSDEM 2012 in Brussels last Saturday, are available on the Ada-Belgium web site now:

- "Welcome & Ada-Europe info"
  by Dirk Craeynest - Ada-Belgium
- "An introduction to Ada 2005 and Ada 2012"
  by Jean-Pierre Rosen - Adalog
- "Ada in the on-line multi-user game Crimeville"
  by Jacob Sparre Andersen - Research & Innovation
- "The contract model of Ada 2012"
  by Jean-Pierre Rosen - Adalog
- "Multicore programming support in Ada"
  by José F. Ruiz - AdaCore
- "Lovelace: towards a full Ada OS"
  by Xavier Grave - Ada-France
- "Programming Arduinos in Ada"
  by Jacob Sparre Andersen - Research & Innovation
- "Programming LEGO MINDSTORMS robots in Ada"
  by José F. Ruiz - AdaCore
- "Ada on Rails"
  by David Sauvage - AdaLabs
- "PPETP: a P2P streaming protocol implemented in Ada"
  by Riccardo Bernardini - University of Udine

Presentation abstracts, copies of slides, speakers bios, pointers to relevant information, links to other sites, etc., are all available on the Ada-Belgium site at

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/12/120204-fosdem.html

We'd like to also put some pictures and recordings online that were taken during the event. If you have material you would like to share, or know someone who does, then please contact me.

Finally, thanks once more to all presenters and helpers for their work and collaboration, thanks to the many participants for their interest, and thanks to everyone for another nice experience!

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Tue, 7 Feb 2012 23:05:10 +0000
Subject: Re: FOSDEM 2012 - Presentations Ada Developer Room on-line
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

Ada seems to have made a good impression at FOSDEM 2012 last weekend.

See among others a brief blog-report on the talk about programming a Lego Segway-like robot : http://legopunk.com/?q=node/104

Also, see some of the reactions on Twitter:

http://twitter.com/#!/search/Ada%20FOSDEM

Was the best FOSDEM moment the Lego Segway programmed in Ada?

One of my FOSDEM takeaway; it's possible to program Arduino using Ada!

Haven't written any Ada for years - interesting to see the latest developments

Will really need to try Ada out… seems like a cool old school language…

Programming Lego Mindstorm in Ada. This is gonna be cool.

Lego Mindstorms and Ada - curious!

at #fosdem, holding an #arduino-like board with an accelerometer running #ada and streaming real-time data to the beamer :)

At FOSDEM, being introduced to #ada. Seems lovely ;)

In a huge lecture hall listening to talk on #ada about free and open source software @fosdem by @AdaCore

All in all, it was nice to see quite some interest from a very diverse public. And although this year's DevRoom had 37% more seats compared to our previous one (81 vs. 59), plus more competition from the +-20 parallel tracks, several of our presentations drew a full room.

Ada is getting "cool"… ;-)

Open Source Days 2012 — Ada related presentations

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: Thu, 23 Feb 2012 21:47:27 +0100
Subject: Ada presentations at Open Source Days in Copenhagen
Newsgroups: comp.lang.ada

There will be (at least) two Ada related presentations at Open Source Days in Copenhagen this year.

+ Programming Arduinos in Ada

  Featuring the AVR-Ada compiler and one or more microcontrollers programmed in Ada.

+ A completely unfair and biased web framework benchmark

  Featuring AWS in a prominent place on the list of the "fastest" web frameworks.

Open Source Days takes place in Copenhagen March 10th and 11th this year. Conference web-site: http://opensourcedays.org/2012/

Ada-related Resources

Ada and JSON

From: Thomas Løcke
Date: Mon, 21 Nov 2011
Subject: Getting down and dirty with Ada and JSON
URL: http://ada-dk.org/2011/11/getting-down-and-dirty-with-ada-and-json/

JSON is a pretty neat data-interchange format. It is lightweight and both easy to read and write. It is also ever so slowly becoming the de facto standard for web-applications, because it is extremely easy to use with Javascript. As a matter of fact, JSON is a subset of the object literal notation of JavaScript so it can be used in the language with no muss or fuss.

But how does JSON fare when coupled with Ada?

Pretty good actually. I ventured into the realm of JSON suspecting I would end up having to write my own parser/generator, but as luck would have it, one of my favorite Ada libraries, GNATColl, turned out to have support for JSON in its latest SVN checkout, and even better: It was pretty good!

I started mucking about with the GNATColl.JSON package, and after an initial failure to make it work (due to me not being able to read properly), I made it fly. And it flew well. It is both intuitive to use, and very clean to read. There's really nothing bad to say about the JSON support in GNATColl, except for perhaps one little thing: It's still very new, so new in fact that at the time of writing, there's no entry for it in the GNATColl manual. This is not a huge issue, but it's worth taking into consideration.

To really drive home the fact that I enjoyed working with the GNATCOLL.JSON package, I wrote a couple of articles and dropped some test code at my GitHub page:

My Ada-DK Wiki article Handling JSON Using GNATColl

My FSFE blog post Ada with as side of JSON

My JSON_Test example @ GitHub

In short: JSON and Ada programming is a good match, so before settling on using XML for your data-interchange needs, perhaps it's worth taking a look at JSON? It is much nimbler and very easy to work with.

[read the article above at

http://wiki.ada-dk.org/index.php/Handling_JSON_Using_GNATColl

http://blogs.fsfe.org/thomaslocke/2011/11/18/ada-with-a-side-of-json/

https://github.com/ThomasLocke/JSON_test
—mp]

Ada in Denmark on Google+

From: Thomas Løcke
Date: Fri, 23 Dec 2011
Subject: Ada in Denmark @ Google+
URL: http://ada-dk.org/2011/12/ada-in-denmark-google/

If you're active on the excellent Google+ network, you might be interested in knowing that you can now follow the Ada in Denmark Google+ page.

[https://plus.google.com/u/0/109362349065401961880 —mp]

Obviously we'll post links to Ada news both here and on the Google+ page, but we also plan on using the page for photos/videos from the open Ada-DK meetings, and hopefully for some fun interaction with other Ada programmers.

It would be pretty awesome if we could build a network of Ada'ists on Google+, so come join the revolution!

Red-black tree with SPARK correctness proofs

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Fri, 24 Feb 2012 09:06:59 -0000
Subject: ANN: SPARK: A red-black tree with correctness proofs
Newsgroups: comp.lang.ada

A SPARK package implementing a red-black tree is now available and correctness proofs have been completed for the code. (SPARK correctness is, of course, partial correctness as there are no proofs of termination of the operations.)

The archive can be downloaded from the Data Structures page at:

http://www.sparksure.com/

The readme for the release is:

http://www.sparksure.com/resources/rb_tree_V0_1_ReadMe.txt

In this version the Ada code is complete and all the mandatory SPARK annotations for information flow analysis are included, but the optional proof annotations within the operations in the package body have been excluded. (I have completed these, but they are not yet in a publishable form.)

The tree elements store a single integer value. A skeleton implementation of an Ordered_Set package is included to show how the tree package can be used to create ordered containers for arbitrary element types. The only additional requirement is for a Key function for the element type, returning an integer value, where equivalent elements are defined as those that have the same value for Key.

If you find this code useful then please let me know (there is an email address with the material in the archive). In particular I am keen to know whether anyone would like to have the annotations and rules that complete the correctness proofs.

Ada-related Tools

Matreshka 0.2.0

From: Vadim Godunko <vgodunko@gmail.com>

Date: Sat, 7 Jan 2012 14:43:41 -0800
Subject: Announce: Matreshka 0.2.0
Newsgroups: comp.lang.ada

We are pleased to announce new major release of Matreshka. New release includes:

- XML Base support
- XML Catalogs support
- IRI/URI utilities
- Windows-1250 and Windows-1252 codecs
- module to access to UML models

Complete list of new features is available

http://forge.ada-ru.org/matreshka/wiki/ReleaseNotes/0.2.0

Source code is available for download

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

[see also "Matreshka 0.1.1" in AUJ 32‑3 (Sep 2011), p.136 —mp]

Tables for Ada v1.11

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 22 Jan 2012 11:24:28 +0100
Subject: ANN: Tables for Ada v1.11
Newsgroups: comp.lang.ada

The library provides an implementation of tables searched using string keys case-sensitive and insensitive. Tables support search for names of unknown length, i.e. to parse a string using the table.

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

The new version provides Fedora and Debian packages for both 32- and 64-bit architectures.

[see also "Tables for Ada v1.11" in AUJ 28-2 (Jun 2007), p.74 —mp]

Simple components for Ada v3.13

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 23 Jan 2012 19:20:03 +0100
Subject: ANN: Simple components v3.13
Newsgroups: comp.lang.ada

The library provides implementations of

- smart pointers and persistency layers backed by a database,
- containers and data structures: directed graphs, sets, maps, stacks, tables, string editing, unbounded arrays,
- expression analyzers and parsers,
- 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, block and storage streams,
- IEEE 754 representations support.

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

New in this release:

- minor bug fixes in the packages Generic_Blackboard and Strings_Edit.Symmetric_Serialization;
- adaptation to 64-bit targets;
- Fedora and Debian packages are provided for both 32- and 64-bit architectures.

[see also "Simple components for Ada v3.12" in AUJ 32-4 (Dec 2011), p.212 —mp]

GTKAda contributions v2.11

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 23 Jan 2012 21:24:15 +0100
Subject: ANN: GtkAda Contributions v2.11
Newsgroups: comp.lang.ada

The library is a contribution to GtkAda. It deals with the following issues:

- Tasking support;
- Custom models for tree view widget;
- Custom cell renderers for tree view widget;
- Multi-columned derived model;
- 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;
- 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

This version is compatible with the GtkAda version 2.24.0.

[see also "GTKAda Contributions v2.10" in AUJ 32-4 (Dec 2011), p.214 —mp]

Paraffin 2.4

From: Brad Moore <brad.moore@shaw.ca>
Date: Sat, 11 Feb 2012 11:17:21 -0700
Subject: ANN: Paraffin 2.4
Newsgroups: comp.lang.ada

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 suite of useful parallel utilities that utilize the Paraffin generics. These include generics for;

1) generic to integrating a function in parallel
2) generic to apply quicksort algorithm in parallel to an array
3) generic to apply fast fourier transform to an array of data.
4) generic Red-Black tree container that performs some operations in parallel.
5) function to solve matrices using Gauss-Jordan Elimination

Paraffin 2.4 modifications include:

- Added a reusable utility to solve a matrix of linear equations using Gauss-Jordan Elimination.
- Added test_matrix test driver for parallel/sequential matrix solvers
- Red-Black Tree containers fully implemented now. (Delete, and Contains calls work)
- Split Red-Black Tree container into separate generics
  - o Sequential
  - o Work sharing
  - o Work Seeking
  - o Stack Safe Work Seeking
- The Red Black Tree generic was previously intended mostly as a test driver for Paraffin Recursive generics. Now the generic has been completed to a state where it can be reused on its own as as generic container.
- Changed use of Unchecked_Conversion for Recurse subprogram access to use 'Unrestricted_Access instead. This cleaned the code up considerably in this area. One issue to look into, is that 'Unrestricted_Access is a non-standard attribute. It is supported by both GNAT and the ICC compiler however. It may not be supported by other Ada 2005 compilers. Unfortunately, Unchecked_Conversion in this case is not portable either, and may not even continue to work with the existing

compilers, so it was thought that using 'Unrestricted_Access was the best option for now. Will investigate to see if there is a possibility for a better solution, or providing a portable mechanism in a future version of Ada.

[download Paraffin from http://sourceforge.net/projects/paraffin/ see also "Paraffin" in AUJ 32-1 (Mar 2011), p.8 —mp]

Interval arithmetic v1.10

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 13 Feb 2012 18:42:36 +0100
Subject: ANN: Interval arithmetic for Ada v1.10
Newsgroups: comp.lang.ada

The package provides an implementation of interval arithmetic.

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

This version is packaged for Fedora and Debian, 32- and 64-bit x86 architectures.

[see also "Interval Arithmetic for Ada v1.8" in AUJ 31-2 (Jun 2010), p.87 —mp]

Visual Ada Developer 7.6

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Mon, 2 Jan 2012 04:05:36 -0800
Subject: Announce: Visual Ada Developer VAD 7.6
Newsgroups: comp.lang.ada

Visual Ada Developer (VAD) 7.6 is now available

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

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

VAD is distributed in the hope, that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

VAD 7.6 Common description.

1. VAD (Visual Ada Developer) is a Tcl/Tk oriented Ada-95(TCL) GUI builder portable to difference platforms, such as Windows NT/Vista/7, Unix(Linux), eComStation(Os/2) and Mac.

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

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

VAD 7.6 was tested in Windows 32bit/64bit and Linux x86 Debian 5

2. Used software

GNAT GPL 2011 Ada-05 compiler (or any others 95, 2005 or 2012)

[complete list of used software stripped —mp]

[see also "Visual Ada Developer 7.4" in AUJ 31-3 (Sep 2010), p.158 —mp]

Ada industrial control widgets v1.1

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 24 Jan 2012 22:02:51 +0100
Subject: ANN: Ada industrial control widgets v1.1
Newsgroups: comp.lang.ada

The library provides means for designing high-quality industrial control widgets for Ada applications. The software 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

This release introduces waveform layers and multi-channel oscilloscope widgets for rendering massive amounts of data in real-time, with data sampled asynchronously to rendering. The oscilloscope widget can be used for plotting purpose as well. The widget supports graph papers, annotated axes, auto- and manual scaling, visual zooming, zooming undo/redo buffers, mouse hovering. All widgets support rendering of snapshots on the surfaces supported by cairo, e.g. into a PDF or SVG file. The library is fullycompatible to the GtkAda versions 2.14, 2.18 and 2.24. It is packaged for Debian and Fedora, 32- and 64-bit x86 platforms.

[see also "Ada industrial control widget library v1.0" in AUJ 32-1 (Mar 2011), p.11 —mp]

GWindows (11-Feb-2012)

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sat, 11 Feb 2012 07:05:15 -0800
Subject: Ann: GWindows installer, 11-Feb-2012
Newsgroups: comp.lang.ada

Hello,

A short announce about a new release of GWindows, a GUI framework for Windows.

For years this framework has been maintained and developed (especially, it builds for both 32 and 64 bit!), but only available through a SVN checkout.

Now there is a setup program, GWindows Setup 11-Feb-2012.exe, that facilitates the access to that fantastic framework. In particular, you can choose easily between ANSI and Unicode flavours.

Look here for download:

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

Note that the setup program is 100% in Ada, and of course made with GWindows on the GUI side.

[see also "GWenerator 0.99" in AUJ 31-1 (Mar 2009), p.11 —mp]

External logging support for AWS

From: Thomas Løcke
Date: Fri, 11 Nov 2011
Subject: Adding external logging support to AWS
URL: http://ada-dk.org/2011/11/adding-external-logging-support-to-aws/

As you know, I like the AWS (Ada Web Server) project a lot, and I have on previous occasions submitted patches to them, when I felt I had a contribution worthwhile bothering the AWS developers with.

This is one such occasion.

Together with a bunch of Ada-DK guys, I've started a new business […], and we plan on using AWS heavily, and one of the small "annoyances" we stumbled on, was the fact that AWS could only log it's activity and error log data to a local file.

This is both potentially inefficient under heavy load, and it is a real security issue if the server is compromised, because the attacker then have easy and unlimited access to the log data.

So we really wanted to enable AWS to send its log data to something like syslogd, and as luck would have it, another AdaCore package could help us do just that: GNATColl. This little marvel of a package can talk to syslogd, so now all that was needed was a way to connect AWS to said functionality.

And that's just what I've done. Or rather, I've written a patch that gives AWS users the ability to send access and/or error log data to an external procedure.

This procedure can then make use of GNATColl's Traces.Syslog package to handle the log data. Obviously this also

opens up the possibility of sending the log data to wherever it might suit a given AWS user. You don't have to use GNATColl.

It's a very flexible and basic system really. My hope is that even if the AWS developers dismiss my patch, for one reason or another, they'll at least consider adding the functionality to AWS, because it really is a deal breaker […] if AWS can't do logging to an external service such as syslogd.

We'll be keeping our fingers crossed.

[find the patch at https://gist.github.com/1348697 —mp]

AWS patched for the 28c3 hash vulnerability

From: Thomas Løcke
Date: Fri, 20 Jan 2012
Subject: AWS patch the 28c3 hash vulnerability
URL: http://ada-dk.org/2012/01/aws-patch-the-28c3-hash-vulnerability/

January the 3rd.and 9th. I posted two short messages to the AWS mailing list, asking whether AWS was susceptible to this attack […]. I got no answer, but from looking at the hash function used, I was pretty sure AWS was just as vulnerable as all the other web technologies. This was confirmed at the January open Ada-DK meeting, where we spent some time checking out the code.

Things were also stirring at the #ada Freenode IRC channel, and January the 17th.

Marcelo Freitas put together a test, and actually found 46656 hash collisions simply by bruteforcing the Ada.Strings.Hash function, which is what AWS used at the time.

Those 46656 collision were enough to keep one core running at 100% for 3 minutes, using a simple "Hello World" AWS server.

Marcelo sent his findings to the AWS developers, and shortly thereafter these fixes were pushed to the AWS Git repository:

Implement a secure string hash routine.

[https://forge.open-do.org/plugins/scmgit/cgi-bin/gitweb.cgi?p=aws/aws.git;a=commitdiff;h=9f1405b79f48cc0c98b97b7c84dee6c1107dae8a —mp]

Use AWS.Utils.Hash secure string hash routines.

[https://forge.open-do.org/plugins/scmgit/cgi-bin/gitweb.cgi?p=aws/aws.git;a=commitdiff;h=d22ec0db402027f87476d197ef02af24784c0faf —mp]

The power of Open Source software is amazing. This fix went in the same day Marcelo had reported the issue. That is just plain awesome.

AWS now ranks proudly among the few web technologies where this problem has been fixed.

Ada Server Faces 0.3.0

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Tue, 14 Feb 2012 23:18:42 +0100
Subject: [Ann]: Ada Server Faces 0.3.0 is available
Newsgroups: comp.lang.ada

Hi all,

Ada Server Faces is a web framework which uses the Java Server Faces design patterns (See JSR 252 and JSR 314).

JSF and ASF use a component-based model for the design and implementation of a web application.

The presentation layer is implemented using XML or XHTML files and the component layer is implemented in Ada 2005 for ASF and in Java for JSF.

A new version of ASF is available which provides:

- New components used in HTML forms (textarea, select, label, hidden);
- New components for the AJAX framework;
- Support for dialog boxes with jQuery UI;
- Pre-defined beans in ASF contexts: param, header;
- A complete set of example and documentation for each tag.

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

[see also "Ada Server Faces" in AUJ 31‑4 (Dec 2010), p.229 —mp]

QtAda 2.7.4

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Sun, 4 Dec 2011 22:45:20 -0800
Subject: I'm pleased to announce QtAda version 2.7.4
Newsgroups: comp.lang.ada

Announce: QtAda version 2.7.4 free edition.

QtAda is Ada-95(05,12) port to Qt4 graphics library Qt version 4.7.3(4.7.4,4.8.0) open source and qt4c.dll(libqt4c.so) built with Microsoft Visual Studio 2010 in Windows and GCC in Linux. Packages were tested with GNAT GPL 2011 Ada compiler in Windows 32bit and 64bit and Linux x86 Debian 5.

It supports GUI, SQL, Multimedia, Web, Network and many others things.

QtAda for Windows and Linux (Unix) is available from

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

[…]

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Thu, 22 Dec 2011 02:49:01 -0800
Subject: Re: I'm pleased to announce QtAda version 2.7.4
Newsgroups: comp.lang.ada

[…]

QtAda 2.7.4 Release 2 is now available. I added QPlugin support, rebuilt shared library in Linux with Qt 4.7.4, added new demos. And good news: Qt 4.7.3 is now ported to eComStation(Os/2) and precompiled binaries are available on ADASTUDIO 2011 DVD.

QtAda free edition is available

http://users1.jabry.com/adastudio/index.htm=l

[see also "QtAda 2.7.0" in AUJ 31‑3 (Sep 2010), p.159 —mp]

Ada binding to the v8 Javascript engine

From: Kylix <likai3g@gmail.com>
Date: Mon, 23 Jan 2012 03:44:18 -0800
Subject: Ann: ada binding to v8(google javascript engine)
Newsgroups: comp.lang.ada

I made a binding to v8 (google javascript engine), with some examples, such as a simple shell.

[…]

[download the software at http://code.google.com/p/v8a/ —mp]

CUDA/Ada version 0.1

From: Reto Buerki <reet@codelabs.ch>
Date: Tue, 14 Feb 2012 10:15:10 +0000
Subject: ANN: CUDA/Ada version 0.1
Newsgroups: comp.lang.ada

We proudly announce the first release of CUDA/Ada, a binding to NVIDIA's CUDA parallel computing platform and programming model.

The project website is at [1], the current release 0.1 can be downloaded from [2].

An article documenting the binding as well as the originating process can be accessed via the project website. Additionally, there is also a presentation about CUDA/Ada.

This project was developed during the course of the master seminar "Program Analysis and Transformation" at the University of Applied Sciences

Rapperswil. We would be glad to get some feedback from the community about the interest in the topic of GPU programming with Ada.

We are currently considering the improvement of CUDA/Ada as part of the next master seminar if somebody finds this project useful.

[…]

[1] - http://www.codelabs.ch/cuda-ada/

[2] - http://www.codelabs.ch/download/

[see also "On CUDA and Ada" in this AUJ issue —mp]

GNAT-AUX updated to GCC 4.6.2

From: John Marino <dragonlace.cla@marino.st>
Date: Sun, 8 Jan 2012
Subject: GNAT AUX Synced with GCC 4.6.2
URL: http://www.dragonlace.net/posts/GNAT_AUX_Synced_with_GCC_4.6.2/

This has been a long time coming, but the upgrade of GNAT-Aux to version 4.6.2 was not straight-forward. First, two more languages were added: Fortran and Objective-C. As a result of adding these two languages and running their test suites, platform misconfigurations were uncovered, and fixing them is a long iterative process.

The second major change is that custom tarballs are no longer used. Instead, the basis for the compiler are official GCC releases and then GNAT-AUX changes are applied as patches on top of that. This has two benefits:

It will be easier to move to GCC 4.7.0 when it is released and it will be easier to isolate patches to send back to FSF to permanently incorporate into the code base.

Both additional languages and patch isolation have been requested, so this important upgrade satisfies those requests.

FreeBSD: PR to upgrade ports submitted: PR 163914

DragonFly, NetBSD, OpenIndiana: Available on pkgsrc-trunk lang/gnat-aux

It was committed after the 2011Q4 freeze was complete, so if one needs it before 2012Q1 branch is released, they'll have to get it from pkgsrc-trunk.

Test results

Test results over 5 languages on 9 platforms are available on this report.

[http://leaf.dragonflybsd.org/~marino/results/gnataux_test_results.pdf —mp]

From an Ada view, all pkgsrc and ports platforms still pass 100%.

Alog 0.4

From: Reto Buerki <reet@codelabs.ch>
Date: Tue, 22 Nov 2011 14:46:35 +0000
Subject: Announce: Alog 0.4 released
Newsgroups: comp.lang.ada

We are proud to announce the release of Alog version 0.4.

Alog is a versatile logging framework for Ada. This release of Alog contains improved syslog support and reworked exception handling among other smaller cleanups.

For further information visit the new Alog project page at

http://www.codelabs.ch/alog.

D_Bus/Ada

From: Reto Buerki <reet@codelabs.ch>
Date: Mon, 5 Dec 2011 11:17:30 +0000
Subject: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

I'm proud to announce the first release of D_Bus/Ada.

The D_Bus/Ada library provides an Ada binding to the D-Bus message bus used for inter-process communication on most modern Linux desktop systems.

D_Bus/Ada supports all but two basic D-Bus types (file descriptor and signature types are not yet implemented) and all container types [1].

The current release focuses on the client side of the D-Bus API but it is also possible to provide D-Bus services written in Ada using the service object interface of D_Bus/Ada.

For further information visit the D_Bus/Ada project page at [2].

[…]

[1] - http://dbus.freedesktop.org/doc/dbus-specification.html#type-system

[2] - http://www.codelabs.ch/dbus-ada

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Mon, 05 Dec 2011 12:40:19 +0100
Subject: Re: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

[…]

> [2] - http://www.codelabs.ch/dbus-ada

At the bottom of the above page, in section “Examples”, there's a

```
with D_Bus.Arguments.Basic;
with D_Bus.Arguments.Containers;
```

Then later

```
pragma Unreferenced
       (D_Bus.Arguments.Basic);
pragma Unreferenced
       (D_Bus.Arguments.Containers);
```

Do you withed both for initialization side effects ?

From: Reto Buerki <reet@codelabs.ch>
Date: Mon, 5 Dec 2011 12:12:54 +0000
Subject: Re: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

[…]

No. This is needed to make the basic and container types known to the D_Bus/Ada type system. D_Bus/Ada uses generic dispatching to create Arguments from low-level D-Bus message arguments.

The example does not use specific argument extensions, that's why both packages are Unreferenced.

E.g. if you remove '**with** D_Bus.Arguments.Containers' in this example you'll get:

```
raised D_BUS.D_BUS_ERROR :
Unknown type code 'a' in message
```

The type system does not know how to deserialize the D-Bus ARRAY(97) type.

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Mon, 05 Dec 2011 12:45:09 +0100
Subject: Re: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

[…]

Also, please, it's always better to show license terms on-page too, not only in sources.

License is often the first thing one may which to check for.

When license terms are exposed on-page (at least by reference), this avoid the need to download all the source just to check for the license.

Actually, the license for this one seems to be GPL, but the above page does not say anything about it.

From: Reto Buerki <reet@codelabs.ch>
Date: Mon, 5 Dec 2011 12:27:53 +0000
Subject: Re: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

[…]

You are certainly right. Thanks for your feedback! The D_Bus/Ada project page now contains a 'Licence' section.

> Actually, the license for this one seems to be GPL, but the above page does not say anything about it.

The licence of D_Bus/Ada is GMGPL.

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon, 5 Dec 2011 21:43:30 +0200
Subject: Re: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

[…]

For those who are wondering what to do with this, here is an example:

http://iki.fi/tero.koskinen/dbus-ada/notify/

In summary:

```
Name_A :
  Arguments.Basic.String_Type :=
        +"ada.notify";
Replaces_ID_A :
  Arguments.Basic.U_Int32_Type :=
        +33;
App_Icon_A :
  Arguments.Basic.String_Type :=
        +"";
Summary_A :
  Arguments.Basic.String_Type :=
        +"Hello";
Body_A :
  Arguments.Basic.String_Type :=
        +"Hello, World from Ada!";
  …
begin

  -- Arguments
  Args.Append (Name_A);
  …
  Result := Connection.Call_Blocking
   (Connection  => Conn,
    Destination =>
          "org.freedesktop.Notifications",
    Path        =>
          "/org/freedesktop/Notifications",
    Iface       =>
          "org.freedesktop.Notifications",
    Method      => "Notify",
    Args        => Args);
```

After running the program, you should see a notification with text "Hello, World from Ada!" on your GNOME/KDE desktop.

[…]

PS. For some reason dbus-ada does not allow me to use/serialize empty arrays, so I fill the arrays with dummy data in my code.

From: Reto Buerki <reet@codelabs.ch>
Date: Tue, 6 Dec 2011 08:02:56 +0000
Subject: Re: Announce: D_Bus/Ada, talk with your desktop in Ada
Newsgroups: comp.lang.ada

[…]

Thanks for this example Tero. With your permission it is now included in D_Bus/Ada. I slightly simplified it, see [1].

[…]

I have to look into this. I'll try to add support for empty containers in the next release.

[1] - http://git.codelabs.ch/?p=dbus-ada.git;a=blob;f=examples/notify/notify.adb

tg – a test driver generator for Ada programs

From: Thomas Løcke
Date: Tue, 27 Dec 2012
Subject: tg – a Test driver Generator for Ada programs
URL: http://ada-dk.org/2011/12/tg-a-test-driver-generator-for-ada-programs/

Testing software is usually not considered the most fun part of programming.

For most it's a chore, something that we do because we have to, not because we think it's fun and exciting, so naturally if a tool claims to ease this pain, it's worth taking a look at. tg is one such tool:

>tg is a program that helps testing software. If you want to test a piece of software, that normally means you have to execute it over and over again, passing various sets of input data to it, and verifying that it gives the correct results for each input. You normally write a program that does all this automatically. Such a program is called a "test driver".

>tg can generate such test driver programs, given a very succinct description of the individual test cases. tg translates this description into a complete Ada program. If you compile this test driver, link it with the software you want to test, and execute it, it performs all the test cases and tells you whether the software under test behaved as expected or not.

Sounds good, yes? And it gets better when you read the documentation and see how simple it is to get going. Check out this small example or if you're in the mood for some more reading, how about this larger example.

tg is written in Ada 95, and it should compile "out of the box" with the GNAT compiler. tg is the brainchild of André Spiegel.

[http://www.free-software-consulting.com/projects/tg/ —mp]

A comparison between the Aunit and Ahven testing frameworks

From: Thomas Løcke
Date: Wed, 4 Jan 2012
Subject: Aunit vs Ahven
URL: http://ada-dk.org/2012/01/aunit-vs-ahven/

I'll be honest with you: I'm not very good at using testing frameworks.

I know I'm supposed to, but I just can't get around to actually doing it.

Luckily for me, not all programmers are like that, and one of those bright ones is Stephane Carrez of Java 2 Ada fame. [http://blog.vacs.fr/index.php? —mp]

His latest article is named Aunit vs. Ahven and as the name implies, it deals with the two testing frameworks Aunit and Ahven.

> AUnit and Ahven are two testing frameworks for Ada. Both of them are inspired from the well known JUnit Java framework. Having some issues with the Aunit testing framework, I wanted to explore the use of Ahven. This article gives some comparison elements between the two unit test frameworks. I do not claim to list all the differences since both frameworks are excellent.
>
> Writing a unit test is equally simple in both frameworks. They however have some differences that may not be visible at the first glance.

Stephane list both good and bad points for both frameworks and ends up concluding that you don't have to choose between the two! Instead you can use his Util.XUnit package to expose a common interface to both frameworks.

Here's the specification for Util.XUnit for Aunit

[http://code.google.com/p/ada-util/source/browse/trunk/testutil/aunit/util-xunit.ads —mp]

and here is Util.XUnit for Ahven

[http://code.google.com/p/ada-util/source/browse/trunk/testutil/ahven/util-xunit.ads —mp]

Quite nifty.

[read the original post at http://blog.vacs.fr/index.php?post/2011/11/27/Aunit-vs-Ahven —mp]

Ada Utility Library 1.4.0

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Sun, 15 Jan 2012 22:41:33 +0100
Subject: [Ann]: Ada Utility Library 1.4.0 is available
Newsgroups: comp.lang.ada

Hi all,

Ada Utility Library is a collection of utility packages for Ada 2005.

It includes:

- A logging framework close to Java log4j framework,
- Support for properties
- A serialization/deserialization framework for XML, JSON, CSV
- Ada beans framework
- Encoding/decoding framework (Base16, Base64, SHA, HMAC-SHA)
- A composing stream framework (raw, files, buffers, pipes)
- Several concurrency tools (reference counters, counters, pools)

A new version is available which provides:

- Support for localized date format,
- Support for process creation and pipe streams (on Unix and Windows),
- Support for CSV in the serialization framework,
- Integratation of Ahven 2.1 for the unit tests (activate with --enable-ahven),
- A tool to generate perfect hash function

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/.

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Mon, 16 Jan 2012 00:10:38 -0800
Subject: Re: : Ada Utility Library 1.4.0 is available
Newsgroups: comp.lang.ada

Hm, I looked at your beans, and I do not see the value of this.

What's the advantage of returning components with string names versus returning them as usual?

First of all, the components are visible.

[…]

From: Georg Bauhaus
Date: Mon, 16 Jan 2012 12:48:15 +0100
Subject: Re: : Ada Utility Library 1.4.0 is available
Newsgroups: comp.lang.ada

[…]

TTBOMK, beans at the most basic level will usually exist so that there is a way for some kind of configurable framework to automatically write ("serialize") plain old Ada objects to external storage, and restore them as Ada objects, as needed. (There is more, like transactions, or events sent to beans.) The external storage typically provides for storing conventional database types; the storage can be configured without touching the programs, and can be anything that meets some simple requirements, not just RDMSs. Could be JSON, or XML, too. JSON is particularly hip if you want communication with Google terminals, such as Android devices or browser based virtual machines, JSON being a Javascript format (assoc lists).

The easiest way to map plain old objects to external storage is via symbolic names understood at both the Ada end and at the data storage end. The resulting objects can be used by programs written in other languages, too. As long as the data items are complete (no references, say), the data storage needs only know, for each data item to be handled, a pair consisting of the name of data item, and the predefined type of item, perhaps in some data nest fashion. Hence, I think,

```
subtype Polytype is
   Util.Beans.Objects.Object;
-- for illustrating the multi-type nature
overriding
  function Get_Value (
       From : Compute_Bean;
       Name : String) return Polytype;
```

together with the definition of a mapping between Polytype, names, and program types.

<Side-note related="somewhat">Rumor has it that not all projects are entirely happy after having moved away from plain old relational SQL (such as JDBC) to object storage (such as Hibernate).</>

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Mon, 16 Jan 2012 21:31:25 +0100
Subject: Re: : Ada Utility Library 1.4.0 is available
Newsgroups: comp.lang.ada

[…]

The purpose of these Ada beans is to be able to give access to object values and object methods from withing a presentation layer.

The benefit of Ada beans comes when you need to get a value or invoke a method on an object but you don't know at compile time the object or method.

That step being done later through some external configuration or presentation file.

In this case, you need somehow to dynamically retrieve the members of an object (hence the Get_Value) and see what methods it exposes (hence the Get_Method_Bindings).

In a presentation page, the Ada bean values could be referenced as follows:

```
Height: #{compute.height}
Radius: #{compute.radius}
```

which triggers a call to Get_Value with the 'height' or 'radius' names on an object registered under the name 'compute'.

To learn more on this presentation topic, I invite you to look at the following article:

http://blog.vacs.fr/index.php?post/2011/03/21/Ada-Server-Faces-Application-Example

Units of measurement for Ada v3.2

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 27 Jan 2012 18:33:52 +0100
Subject: ANN: Units of measurement for Ada v3.2
Newsgroups: comp.lang.ada

The library is provided for handling dimensioned values in Ada. The library supports irregular and shifted measurement units. String formatting and GTK+ widgets and cell renderers are provided.

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

Changes to the version 3.1:

- The procedure Put in Measures_UTF8_Edit has additional parameters Field, Justify, Fill;
- Fedora and Debian packages are provided for both 32- and 64-bit architectures.

[see also "Units of Measurement for Ada v3.1" in AUJ 31‑3 (Sep 2010), p.162 —mp]

SparForte 1.3

From: Master of Magic <koburtch@gmail.com>
Date: Sat, 4 Feb 2012 14:37:32 -0800
Subject: ANN: SparForte 1.3 (Release Candidate)
Newsgroups: comp.lang.ada

SparForte 1.3 (Release Candidate)

Type : Programming Language

Platforms: Linux i386/x86_64/Alpha and FreeBSD

License: GPL

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

Source URL: https://github.com/kburtch/SparForte

SparForte is an Ada-based command shell, template engine and scripting language. There are 23 built-in packages and more than 80 example scripts.

The sources on GitHub should be considered a release candidate for version 1.3. If you are interested, please try it out and notify me of any bugs you find.

Changes in Version 1.3

- with separate (that is, include files)
- new memcache packages
- new JSON functions and related pragmas
- support for unit testing

See ChangeLog in the sources for a complete list.

Started in 2001 as the Business Shell, SparForte is an open project project. Because I'm working on this in my spare time, let me know if you find SparForte useful. Volunteers are encouraged to contribute examples, tutorials, new built-in packages, post examples to Rosetta Code, etc.

From: Alex R. Mosteo <alejandro@mosteo.com>
Date: Tue, 07 Feb 2012 17:01:39 +0100
Subject: Re: ANN: SparForte 1.3 (Release Candidate)
Newsgroups: comp.lang.ada

[…]

FYI, I managed to compile it without much trouble in Ubuntu 11.10. I had only to install the relevant SDL/mysql/pgsql packages and export GMAKE=make

[…]

From: Julian Leyh <julian@vgai.de>
Date: Thu, 9 Feb 2012 01:17:53 -0800
Subject: Re: ANN: SparForte 1.3 (Release Candidate)
Newsgroups: comp.lang.ada

FYI, I created an ArchLinux PKGBUILD of version 1.2.1 (will update as soon as 1.3 is released):

https://aur.archlinux.org/packages.php?ID=56486

I removed the sound support, because it requires /dev/dsp, and I was told this belongs to OSS, which blocks sound IO while in use.

If anybody knows better or has ideas how to solve this, please correct me.

(maybe oss as makedepends may work?)

Ada-related Products

AdaCore — Qualification material for GNATcheck and GNATcoverage

From: AdaCore Press Center
Date: Tue, 29 Nov 2011
Subject: Qualification Material Available for GNATcheck and GNATcoverage
URL: http://www.adacore.com/2011/11/29/qualification-gnatcheck-gnatcoverage/

AdaCore releases components allowing more agile software certification

NEW YORK, PARIS and TOULOUSE, France, November 29, 2011 – Certification Together Conference – AdaCore, provider of Ada tools and expertise for the mission-critical, safety-critical, and security-critical software communities, today announced the availability of qualification material for two tools: GNATcheck, an Ada coding standard and rule checker; and GNATcoverage, a non-intrusive structural code coverage analyzer. These qualification documents extend existing AdaCore certification material that includes the Traceability Analysis Package, a source-to-object code traceability study for the GNAT Pro High-Integrity Edition. These new products will help the development of certified applications compliant with the DO-178B avionics software safety standard, up to level A, and will apply equally to DO-178C, the upcoming revision to DO-178B.

The qualification packages allow developers to take credit for the use of the GNATcheck and GNATcoverage tools in the certification of applications in accordance with the DO-178B standard, at level A and below (Table A5 objective 4, and Table A7 objectives 5, 6, and 7, respectively).

The Traceability Analysis Package answers the need for the additional verification work required by DO-178B, level A, as part of the structural code coverage activity. It enables the use of the GNAT Pro compiler to meet Table A7 objective 7, in accordance with the guidelines described in the Certification Authority Software Team's Position Paper CAST-12. The traceability analysis material establishes traceability between source code and object code and provides additional verification for untraceable code, as described in section 6.4.4.2.b of the DO-178B standard.

The analysis is performed on a set of code patterns that is representative of the customer's application.

With this release, AdaCore has taken the first steps in developing a certification artifacts management system. Named the "Qualifying Machine," it is an agile framework that supports the development, maintenance and modification of software tools and their associated qualification material.

It facilitates providing multiple versions of a tool, each supported by its accurate and up-to-date qualification material. Modifying the tool to integrate new features, or fixing reported issues, is now possible, and requalification can be done incrementally and in a cost-effective manner. The ultimate goal is to fully support an agile, incremental, and continuous certification process that automates the most time-consuming certification activities, such as the management and verification of traceability data.

This strategy was described in a recent EE Times article 'The "Big Thaw" – An Agile Process for Software Certification.'

"Offering off-the-shelf qualifiable tools that evolve and improve along with the rest of the technology is the challenge we are in the process of solving.

This will bring the latest and most innovative features to those who need them, without compromising the absolute requirement for safety," said Cyrille Comar, President, AdaCore Europe.

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial software solutions for Ada, a state-of-the-art programming language designed for large, long-lived applications where safety, security, and reliability are critical. AdaCore's flagship product is the GNAT Pro development environment, which comes with expert on-line support and is available on more platforms than any other Ada technology. AdaCore has an extensive world-wide customer base.

See http://www.adacore.com/home/company/customers for further information.

Ada and GNAT Pro see a growing usage in high-integrity and safety-certified applications, including commercial aircraft avionics, military systems, air traffic management/control, railway systems, and medical devices, and in security-sensitive domains such as financial services. The SPARK Pro toolset, available from AdaCore, is especially useful in such contexts.

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

[…]

Adalog — AdaControl 1.13r8

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 01 Dec 2011 13:09:14 +0100
Subject: [Ann] AdaControl 1.13r8 released
Newsgroups: comp.lang.ada

A new version of AdaControl is available from

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

Apart from a number of new rules (412 checks!) and bug fixes as usual, this version is now able to process Ada 2005 code without choking. This required updating a number of rules (for example, all rules dealing with return must now handle the extended return statement). A couple of the new rules deal with Ada2005 constructs, too.

Note that, as announced before, the files are now hosted on SourceForge, and that the bleeding edge version is available there from the GIT repository. Note also that it is possible to report issues through the MantisBT system on SourceForge.

If you are a regular user, please click on "I use this" from AdaControl's home page, and/or support it from the SourceForge page.

Let's build up a community of users!

[see also "AdaLog — AdaControl 1.12r3" in AUJ 31-3 (Sep 2010), p.163 —mp]

Ada and GNU/Linux

GNAT 4.6 moving to Debian/testing

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Fri, 17 Feb 2012 09:42:47 +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/2011/12/msg00013.html

adabrowse	OK
adacgi	OK
adacontrol	OK
adasockets	OK
ahven	OK
apq	
apq-postgresql	
asis	OK
gnade	OK
gnat	OK
gnat-gps	OK
gnatpython	OK
(renamed to python-gnatpython)	
gprbuild	OK
libaws	OK
libalog	
libaunit	OK
libflorist	OK
libgmpada	OK
libgtkada2	OK
(renamed to libgtkada)	
liblog4ada	OK
libncursesada	OK
libtemplates-parser	OK
libtexttools	OK
libxmlada	OK
libxmlezout	OK
music123	OK
narval	
(ready but build-depends on polyorb)	
opentoken	OK
pcscada	OK
polyorb	
(ready but build-depends on gnatpython)	
spark	OK
topal	OK

As you can see, all but a few packages have now transitioned to gnat-4.6. In a couple of weeks, I shall request that packages that still have not made the transition be removed from testing (but not unstable) so that other packages can migrate to testing.

Also, a minor change in Debian Policy for Ada is forthcoming: we will remove the virtual package ada-compiler, which is not really necessary and causes lots of lintian warnings [1]. So, please go ahead and upload updates to your packages to remove dependencies on ada-compiler. The last step will be to upload a gnat-4.6 that does not Provide: ada-compiler. I hope to do this last upload in May at the latest, in time for the freeze in June.

[1] http://lists.debian.org/debian-ada/2011/12/msg00012.html

Ada and Java

JVM-GNAT GPL 2011 for Mac OS X Snow Leopard

From: Blady <p.p11@orange.fr>
Date: Sun, 4 Dec 2011 13:14:07 -0800
Subject: JVM-GNAT GPL 2011 binaries for Mac OS X available
Newsgroups: comp.lang.ada

Hi, I've upload JVM-GNAT GPL 2011 binaries for Mac OS X Snow Leopard on Source Forge:

http://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2011-snow-leopard/

In addition to JVM-GNAT from AdaCore, to build the library and tools, some patches have been needed from Stephe Leake (nice from him to put on his web site http://www.stephe-leake.org/)

All seems to be fine but not with Applet, I've got an error:

java.lang.ClassNotFoundException: Applet

Has any one some clue?

Is it due to wrong execution configuration?

Is it due to wrong compilation?

A simple Ada Applet example can be found here:

http://blady.pagesperso-orange.fr/telechargements/jgnat/Essai_Ada.zip

Ada Inside

'Project P' and 'Hi-MoCo' research projects

From: AdaCore Press Center
Date: Wed, 1 Feb 2012
Subject: 'Project P' and 'Hi-MoCo' Research Projects Launched
URL: http://www.adacore.com/2012/02/01/project-p-and-hi-moco/

Open-source research projects combine model-based integration and qualified code generation for safety-critical systems

TOULOUSE, France, PARIS and NEW YORK, February 1, 2012 – ERTS Congress - AdaCore today announced its participation in 'Project P' and 'Hi-MoCo' (High-Integrity Model Compiler), two open-source research efforts supported and partly funded by the French and Estonian national governments and the European EUREKA agency. The combined projects, which started in October 2011, aim to provide an open-source, tunable and qualifiable code generation framework for domain-specific modeling languages. The key idea is to allow control engineers (using Simulink, Stateflow and Scicos/XCos), system engineers (using SysML/MARTE and AADL), and software engineers (using UML) to easily collaborate for system-level model integration, verification, and final optimized code generation targeting the Ada 2012, C/C++ and VHDL languages.

AdaCore, the technical coordinator of the projects, is working closely with the IB Krates team led by Tõnu Näks and members from IRIT (Institut de Recherche en Informatique de Toulouse) led by Marc Pantel, the principal architects of the ITEA GeneAuto project and technology on which Project P and Hi-MoCo are based.

AdaCore, IB Krates, and IRIT will be the principal contributors to the code generation technology at the heart of the toolset being developed. Together with Frédéric Pothon of ACG Solutions and chair of the Tool Qualification subgroup of the DO-178C committee, the three companies will play a major role in the cross-domain qualification effort spanning the avionics, space, and automotive domains. AdaCore's expertise in developing and supporting the certification/qualification of tunable, open-source commercial-of-the-shelf (COTS) components and tools that meet safety and reliability certification standards such as DO-178 (avionics) and ECSS-E-ST-40 (space) will be especially relevant.

"The ultimate goal of these projects is to end the segregation between the control, system and software engineers," said Franco Gasperoni, Managing Director of AdaCore. "A major bottleneck in the model-driven development of software for avionics, space, and automotive systems is the integration of heterogeneous models and the lack of comprehensive verification and code generation technologies. Project P and Hi-MoCo aim to solve this problem by developing an open-source, tunable and qualifiable code generation framework for heterogeneous models, while making cross-domain qualification material available."

"The current state-of-the-art is to perform integration on generated sources," concluded Matteo Bordin, project manager of the Project P and Hi-MoCo efforts at AdaCore. "We are proposing to do this at the model level to verify integration issues well before models are mature enough for code generation."

About Project P

Project P is a three-year research project financially supported within the French FUI 2011 funding framework. Headed by Continental Automotive France, it involves the collaboration of 19 partners, including major industrial users from the avionics, automotive and space domains (Airbus, Astrium, Continental Automotive, Rockwell Collins, Safran, Thales Alenia Space and Thales Avionics), technology providers (AdaCore, Altair, STInformatique, Scilab Enterprise), service companies (ACG Solutions, Aboard Engineering, Atos Origins) and research centers (ENPC, IRIT-INPT/CNRS, INRIA, ONERA, Lab-STICC/Université de Bretagne Sud). Additional information can be found at

http://www.open-do.org/projects/p.

About Hi-MoCo

Hi-MoCo is a two-year research project financially supported within the Eurostar 2011 funding framework. It supports the collaboration of IB Krates (Estonia), IRIT, and AdaCore.

[…]

Eurocopter selects GNAT Pro for Military Helicopter ARINC 653 Project

From: AdaCore Press Center
Date: Tue, 29 Nov 2011
Subject: Eurocopter Selects GNAT Pro for Military Helicopter ARINC 653 Project
URL: http://www.adacore.com/2011/11/29/arinc-653-project/

NEW YORK, PARIS and TOULOUSE, France, November 29, 2011 – Certification Together Conference – AdaCore, provider of Ada tools and expertise for the mission-critical, safety-critical, and security-critical software communities, today announced that Eurocopter has chosen the GNAT Pro High-Integrity Edition for development of an ARINC-653 demonstrator for military helicopters.

The demonstrator will provide military interfaces and operational functions within a time- and memory-partitioned ARINC-653 architecture.

The GNAT Pro High-Integrity Edition for DO-178B will be used to port military avionics operational functions and Ada software drivers onto an ARINC-653 platform. The objectives of the project are twofold: to demonstrate Integrated Modular Avionics (IMA) capabilities at the test rig level in the military domain, and to provide an ARINC-653 platform to capture technical and process requirements for following IMA military projects.

The GNAT Pro High-Integrity Edition for DO-178B is an enhanced version of the GNAT Pro Ada development environment, designed for building safe and secure software. In addition to some of its toolchain features, specifically developed for the highest levels of safety, it includes qualifiable tools (coding standard checker, static stack size analyzer) that help reduce the cost of developing and certifying systems that have to meet safety standards such as DO-178B. Key to achieving this goal is the product's fully configurable and customizable run-time library. Units can be selected in an a la carte fashion, thus limiting the run-time library to just those units that are required for the Ada features used in the application, while also making it possible to adapt their implementation, if desired.

[…]

About Eurocopter

Established in 1992, the Franco-German-Spanish Eurocopter Group is a division of EADS, a world leader in aerospace, defense and related services. The Eurocopter Group employs approximately 17,500 people. In 2010, Eurocopter confirmed its position as the world's number one helicopter manufacturer in the civil and parapublic market with a turnover of 4.8 billion Euros, orders for 346 new helicopters and a 49 percent market share in the civil and parapublic sectors. Overall, the Group's helicopters account for 33 percent of the total worldwide civil and parapublic fleet. Eurocopter's strong worldwide presence is ensured by its 30 subsidiaries and participations on five continents, along with a dense network of distributors, certified agents and maintenance centers. There are currently 11,200 Eurocopter helicopters in service and some 2,900 customers in 147 countries. Eurocopter offers the largest civil and military helicopter range in the world.

Indirect information on Ada usage

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

Job offer [France]: Embedded Software Engineer

In the department for the development and maintenance of embedded software for on-board systems […] (Ariane 5, Vega, ATV, or other demonstrators), you will work in the development team.

Tasks and responsibilities

You will participate to the following activities:

- Specification and formalization of requirements;
- Design, implementation and test of products;
- Project reviews of software for the aforementioned products.

Required skills

You hold an Engineering MSc with a specialization in Embedded Real-Time Systems, and you have previous experience with the following:

- Unix environment
- C, C++ and Ada programming languages
- Advanced software design methods (UML/SysML, SCADE, formal methods, model-based engineering)
- Run-time execution profiles (C-SMART, T-SMART, etc.)

[…]

[translated from French —mp]

Job offer [France]: Embedded Software Engineer

[…]

You will work in a project team for the development of future IMA (Integrated Modular Avionics) systems […]

You will be in charge of developing safety- and security-critical applications using a V life-cycle (design, implementation and unit tests) and use methodologies that conforms with the constraints of the domain (DO-178B standard).

You have a scientific 5-year degree and you have previous experience in the development of safety-critical software (embedded or real-time) and in adhering to development standards.

You have good knowledge of one of the following languages: C/C++/Ada

The knowledge of the specification language B or of modeling with SCADE is an asset.

Your professional skills and your contribution to the project will enable the progress of your career towards Project Lead or Senior Expert positions.

[…]

[translated from French —mp]

Job offer [France]: Embedded Software Architect

[…]

You will work in a site specialized in the design and production of electronic on-board and ground systems (monitoring & control, signaling).

The activity concerns the definition and conception of electronic and telecommunication solutions for future railway systems.

Your task:

Working in the "Software Architecture" team of the Engineering department, you will participate in the development of our electronic systems.

Considering our client requirements:

- You will define the specification and the software architecture of the product;
- You will lead the development;
- You will have the role of senior expert among the development teams.

Your profile:

You are an engineer in industrial informatics, and you have at least 3-years experience in the design and modeling of systems with UML or SADT/SART. You master RTOS such as Linux, QNX or µC/OS, the languages C or Ada, and you are knowledgeable of open software.

You have good experience in the development of critical systems.

[…]

[translated from French —mp]

Job offer [United Kingdom]: Software Engineer

[…]

The On Board Software Group require a hardworking and enthusiastic engineer who will be responsible for participating in Technology Research and Development activities for future missions. This will involve:

- Contributing innovative ideas to improve future spacecraft software development projects.
- Developing business opportunities with internal and external technical staff.
- Developing software to flight or demonstrator standards as required
- Producing requirements and design documentation to ESA [European Space Agency —mp] standards
- Preparing and presenting papers at technical conferences

Essential skills:

-The ideal candidate must have an engineering, computer science or scientific degree (min 2.1)

Specific knowledge is required in:

- Real-time software systems and computer science
- Modern software engineering methods and tools

Experience is required in:

- Programming languages C or Ada and Java
- Object-oriented design methods
- Embedded systems and development
- Real-time operating systems

Desirable skills:

- SPARC microprocessors
- Communication systems protocols
- UML
- Hardware design and development e.g. use of VHDL
- ESA space programmes and applications
- Software development standards (preferably ECSS-E40)
- Proposal Preparation
- Giving Technical presentations

[…]

Job offer [United Kingdom]: Software Engineer

[…] seeking experienced Software Engineers to join an established team, who are responsible for upgrading simulation software used to train Royal Navy Maritime Operators in a "synthetic environment".

Due to expanding workload for equipment upgrades, additional Software Engineers are required on the project. […]

Job Details

- To monitor, assess and produce innovative real-time computing product concepts, demonstrators, standards and specifications, both bespoke and 'off the-shelf'.
- To evolve from the software requirements a design and test philosophy in order to produce a top level and detailed design that will be used as a baseline for the maintainable software implementation including the prediction of software size and performance.
- To produce unit and integration test descriptions that incorporate the appropriate test philosophies.
- To produce fully tested code units in the appropriate software language that fully implement all requirements on the Client hardware, using the software detailed design as the baseline.

Responsibilities

Reporting to the Software Development Lead, the successful candidates will be part of the team responsible for the upgrade of a number of simulations.

The role will include:

- Requirements analysis and definition
- Software Design and Development
- Software Implementation using a variety of languages including C, C#, Ada 83/95.
- Integration and Acceptance
- Supporting site acceptance activities
- Configuration Management and Software Build (Subversion)
- Customer liaison
- Defect Management

Job offer [United Kingdom]: Software Engineer

Software Engineer - War Game Simulation Key Responsibilities:

- To support Requirements Capture, Functional Specification and Software Design
- Development of software using native languages or other software packages
- Design test procedures, and perform Verification and Test of software
- Provision of software consultancy and advice to clients
- Perform and deliver allocated project tasks, to quality, time and budget requirements, as directed by project manager and/or team leader
- Diligent and accurate work ethic, recognising that many of the applications we work on are safety-related
- Communicate effectively with colleagues and clients.

Software Engineer - War Game Simulation Qualifications and Experience:

- Numerate degree (Computer Science or Mathematics preferred)
- Good Knowledge of one or more of the following programming languages: C, C++, C#, Ada, Visual Basic or Java in the Visual Studio .NET environment
- Experience of PC based software development
- Enthusiastic proponent of full software life-cycle best practice (requirements capture, design, documentation, testing, etc.)
- Experience of war gaming or simulation model development
- Experience of database development using SQL Server
- Experience of GUI development
- Experience of safety-related software
- Experience of embedded, real time systems
- We will also be interested in hearing from any candidates with experience of Formal Methods or Compiler Development.

Ada in Context

On CUDA and Ada

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Fri, 16 Dec 2011 07:05:53 +0000
Subject: NVIDIA opens up CUDA compiler for other languages
Newsgroups: comp.lang.ada

Just read the following announcement:

<quote>

NVIDIA Opens Up CUDA Compiler

GPU maker NVIDIA is going to make its CUDA compiler runtime source code, and internal representation format public, opening up the technology for different programming languages and processor architectures. […]

Read More: http://www.hpcwire.com/ct/uz5609392Biz12256180

</quote>

The full announcement says the LLVM compiler infrastructure will be used as the vehicle for the public CUDA source code. The main idea seems to be to allow others to use the technology (among others) with programming languages that NVIDIA is not pursuing themselves.

Interesting for Ada?

From: Adrian-Ken Rueegsegger <ken@codelabs.ch>
Date: Fri, 16 Dec 2011 14:43:56 +0100
Subject: Re: NVIDIA opens up CUDA compiler for other languages

Newsgroups: comp.lang.ada

[…]

If I understood the article correctly this requires a working LLVM-Frontend for Ada. I do not know what the status of the current implementation [1] is but maybe somebody more knowledgeable about LLVM with regards to Ada could comment on that and give their take?

Incidentally Reto Buerki and me have been working on an Ada-Binding for CUDA. Those interested in using CUDA from Ada might want to check out the project website [2] and the source repository [3].

The low-level thin-binding is complete and we added some abstractions on top of that. Many ideas/goals were inspired by the excellent Python binding pyCUDA [4]. We have not made a release yet since we are currently working on the documentation. If you give CUDA/Ada a try we would be happy if you would let us know what you think.

[…]

[1] - http://llvm.org/docs/GCCFEBuildInstrs.html#ada

[2] - http://www.codelabs.ch/cuda-ada/

[3] - http://git.codelabs.ch/?p=cuda-ada.git

[4] - http://mathema.tician.de/software/pycuda

[see also " CUDA/Ada version 0.1" in this AUJ issue —lmp]

From: jonathan <johnscpg@googlemail.com>
Date: Fri, 16 Dec 2011 07:08:47 -0800
Subject: Re: NVIDIA opens up CUDA compiler for other languages
Newsgroups: comp.lang.ada

[…]

I haven't had a chance to try it yet, but the new LLVM 3.0 has been out a few weeks, along with the new Ada front end

http://dragonegg.llvm.org/

They say:

> Patching and building GCC is no longer required: the plugin should work with your system GCC (version 4.5 or 4.6; on Debian/Ubuntu systems the gcc-4.5-plugin-dev or gcc-4.6-plugin-dev package is also needed).

which sounds encouraging.

From: Rugxulo <rugxulo@gmail.com>
Date: Mon, 19 Dec 2011 15:35:34 -0800
Subject: Re: NVIDIA opens up CUDA compiler for other languages
Newsgroups: comp.lang.ada

[…]

Clang does not have an Ada frontend. It's a "C language"-based compiler only, e.g. C / C++ / Objective C / Objective C++. LLVM is the backend, hence the (now deprecated, no longer updated, not available in LLVM 3.0) LLVM-GCC (GCC 4.2) compiler was needed to compile Ada source code while targeting the LLVM backend.

DragonEgg is a plugin for GCC proper (specifically, 4.5.x or 4.6.x) which targets the LLVM backend. Since GCC (only, and not Clang) has an Ada frontend / compiler, you must use that.

Raspberry PI and Ada

From: Simon Wright <simon@pushface.org>
Date: Tue, 10 Jan 2012 12:16:06 +0000
Subject: Raspberry Pi
Newsgroups: comp.lang.ada

This Armv6-based linux-on-a-credit-card [1] for $25 ($35 with Ethernet) is going to be available soon, with Debian (amongst others).

I see that Debian stable […] has armel support for GNAT and emacs but not GPS.

Sounds hopeful?!

[1] http://www.raspberrypi.org/

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 10 Jan 2012 06:24:21 -0800
Subject: Re: Raspberry Pi
Newsgroups: comp.lang.ada

[…]

gnat-gps 4.3-5 is available on armel in Debian 6 "Squeeze".

gnat-gps 5.0-4 is available on armel in Debian unstable (but took 9 hours and 20 minutes to build…)

> Sounds hopeful?!

[1] http://www.raspberrypi.org/

Interesting! This looks like something even Richard Stallman would approve of :)

From: Simon Wright <simon@pushface.org>
Date: Tue, 10 Jan 2012 16:23:41 +0000
Subject: Re: Raspberry Pi
Newsgroups: comp.lang.ada

I was taking the info from the bottom of

http://packages.debian.org/stable/devel/gnat-gps which doesn't show armel. But good to know it _is_ available!

> gnat-gps 5.0-4 is available on armel in Debian unstable (but took 9 hours and 20 minutes to build…)

Much easier to see the availability in unstable/devel, thanks (esp. for all that build time! I hope it was unattended…)

On elaboration circularity with generics

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sat, 14 Jan 2012 07:09:18 -0800
Subject: Elaboration circularity with generics
Newsgroups: comp.lang.ada

Consider:

```
-- p.ads:
package P is
  procedure Proc;
end P;

-- p.adb:
with P.Q;
package body P is
  package P_Int is
     new P.Q (T => Integer);
  procedure Proc is
  begin
    P_Int.Proc;
  end Proc;
end P;

-- p-q.ads:
generic
  type T is private;
package P.Q is
  procedure Proc;
end P.Q;

-- p-q.adb:
package body P.Q is
  procedure Proc is
  begin
    null;
  end Proc;
end P.Q;

-- test.adb:
with P;
procedure Test is
begin
  null;
end Test;
```

The idea is that P.Q is a child, helper unit for P and P.Q is used in the body of P.

Ideally it should be a private child, but these cannot be generic (why?).

Forget the Proc procedures, they do not contribute to the actual problem, but where needed to have meaningful source units.

The problem above is:

```
$ gnatmake test
gcc -c test.adb
gcc -c p.adb
gcc -c p-q.adb
gnatbind -x test.ali
error: elaboration circularity detected
info:    "p (body)" must be elaborated
before "p (body)"
```

```
info:      reason: implicit Elaborate_All in
unit "p (body)"
info:      recompile "p (body)" with -
gnatwl for full details
info:        "p (body)"
info:          must be elaborated along
with its spec:
info:        "p (spec)"
info:          which is withed by:
info:        "p.q (spec)"
info:          which is withed by:
info:        "p (body)"

gnatmake: *** bind failed.
```

Why this circular dependency? It does not exist if P.Q is not generic.

My initial version of P.Q was a regular package and I have found this problem after turning it into a generic package.

A simple workaround is to make P_Q instead of P.Q, but I would like understand where the circularity comes from. The -gnatwl option says that some Elaborate_All is introduced at the instantiation of P.Q, but I see no reason for circularity there.

From: Martin Dowie <martin@thedowies.com>
Date: Sat, 14 Jan 2012 10:13:13 -0600
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

I think a better way is to make the instantiation another child, e.g.

```
package P.Q.Integers is
   new P.Q (Integer);
pragma Preelaborate (P.Q.Integers);
```

Then it is available to other packages that may need it too.

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Sat, 14 Jan 2012 08:17:25 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

A unit can only be instantiated if it is fully elaborated.

I guess GNAT chose the elaboration order P'Spec, P.Q'Spec, P'Body (crash since P.Q'body is not elaborated).

This dependence is not present for nongeneric units.

Perhaps addition of pragma Elaborate_Body to P.Q helps.

From: Simon Wright <simon@pushface.org>
Date: Sat, 14 Jan 2012 22:46:31 +0000
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

```
with P.Q;
pragma Elaborate (P.Q);  << does the
                            trick for me
package body P is
```

I have to say that elaboration problems are often mindboggling (the best I remember was when GNAT 3.09 reported an elaboration cycle of 157 where there were only 156 units in the program).

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Sun, 15 Jan 2012 08:55:57 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

Of course this works, but in my opinion, Elaborate_Body is better because it has to be applied just once to P.Q, whereas Elaborate has to be applied on every unit withing P.Q.

As a general rule of thumb I think Elaborate_Body should be applied whenever a unit provides functions that are used in the spec of other units providing initial values or constants, such like:

```
package P is
  pragma Elaborate_Body;  -- prevents
                          -- elaboration error when F is
                          -- called
  function F (…) return T;
end P;

with P;  -- no need for "pragma
                        Elaborate (P);"
package Q is
  V: [constant] T := P.F (…);
end Q;
```

From: Simon Wright <simon@pushface.org>
Date: Mon, 16 Jan 2012 15:09:18 +0000
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

> […] As a general rule of thumb I think Elaborate_Body should be applied whenever a unit provides functions that are used in the spec of other units providing initial values or constants, […]

That may be true in general, but *in this case* it does not solve the problem […]

-gnatE still succeeds (provided you remember to rebuild the world!)

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Mon, 16 Jan 2012 09:15:04 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

Ha, yes, because of the implicit Elaborate_All implied by the nonstandard behaviour of GNAT (I didn't think of this connection).

Obviously with standard Ada behaviour (-gnatE), GNAT can find a good order without the pragma (there is no Elaborate_All in this case).

But I think the rule of thumb should be followed nevertheless. (It's only a rule of thumb because Elaborate_Body is sometimes impossible.)

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Sat, 14 Jan 2012 08:26:59 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

> Ideally it should be a private child, but these cannot be generic (why?).

Of course it can:

```
private generic
package P.Q is
```

[…]

But such a private generic can only be instantiated within the hierarchy of P (of course).

From: Georg Bauhaus
Date: 14 Jan 2012 22:01:32
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

The circularity vanishes with -gnatE, and the warning message changes, too. So I guess the circularity is a consequence of GNAT's default, static elaboration order algorithm. IIRC, some, though not all of this algorithm is explained in the manual (else in the source text).

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sun, 15 Jan 2012 09:15:18 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

Does it mean that the compiler can refuse legal code just because its internal algorithm is biased?

As I understand, there is no language-related reason for the circularity. That is, there is no ARM paragraph that would say that in this particular program there will be a cyclic elaboration dependency (otherwise I would appreciate some pointers).

[…]

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Sun, 15 Jan 2012 09:43:34 -0800
Subject: Re: Elaboration circularity with generics

Newsgroups: comp.lang.ada

[…]

No, this isn't a bug. The elaboration order is not completely defined.

If the compiler cannot find a valid order, it may reject the program.

The programmer then has to help the compiler with pragmas.

A compiler is not required to try out all sorts of elaboration orders.

This is for simplifying the compiler. There are about O(n!) orders.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Mon, 16 Jan 2012 09:34:44 -0500
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

By default, GNAT uses a static elaboration model. This is a non-standard mode -- it does not follow the rules in the Ada RM.

If you want the standard mode, you have to turn it on explicitly.

I suggest you read the section about elaboration in the GNAT docs -- it's rather long, but it should explain everything.

One advantage of the static elaboration model is that you don't get any run-time elaboration checks. Another is that you don't need to put elaboration control pragmas all over the place. (On rare occasions, you might need one.) But the static elaboration model is (necessarily) more restrictive in that it will disallow certain cases that are allowed by the standard (dynamic) model.

[…]

From: Adam Beneschan <adam@irvine.com>
Date: Mon, 16 Jan 2012 09:02:13 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

I think some clarification may help. There's a rule in 10.2(18) that says "there shall be a total order of the library_items that obeys the above rules". This rule is simple enough that any compiler should be able to follow it; so if there is a total order and a compiler cannot find it, the compiler definitely has a bug.

However, the compiler is *not* required to try to find an elaboration order that will guarantee that Program_Error is not raised at runtime. Technically, a program that follows the language rules but is certain to fail as soon as it's run should not be rejected by the compiler, since it's a legal Ada program. But it's useful for compilers to report errors in such cases. This may make it "non-standard" in some sense, but I wouldn't consider it a bug if the behavior is documented and an option is provided to follow the standard technically.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Mon, 16 Jan 2012 13:29:54 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

What I'm concerned about is portability. If the compiler is allowed to refuse my program even though there is no paragraph saying that my program is illegal, then perhaps some other compiler will compile it without any trouble. Which means that my program will not be portable, even though it might not touch any implementation limits or other similarly valid reasons.

From: Adam Beneschan <adam@irvine.com>
Date: Mon, 16 Jan 2012 13:52:34 -0800
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

[…]

The problem is that without any additional Elaborate pragmas, a compiler *could* compile the program, but it would be useless as it would raise Program_Error right away. Another compiler might build the program and choose a different elaboration order that doesn't raise Program_Error. Neither of these compilers would be wrong, since the language doesn't specify the exact elaboration order when there is more than one that obeys the Elaborate pragmas and other language rules. So your program as written isn't portable at all; and even though technically a compiler should accept the program, if you're concerned about portability then you should be grateful that GNAT (in its non-standard mode) rejected the program and alerted you to the portability problem in your code. I don't think you have a legitimate complaint about GNAT here, at least not on portability grounds.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Mon, 16 Jan 2012 17:25:03 -0500
Subject: Re: Elaboration circularity with generics
Newsgroups: comp.lang.ada

> […] If the compiler is allowed to refuse my program even though there is no paragraph saying that my program is illegal,…

That's not the problem. If your program is legal (I didn't look at it carefully), then all Ada compilers will accept it. In particular, GNAT will accept it in standard-conforming mode. You didn't use the standard-conforming mode.

The problem (a language problem) is that in standard-conforming mode, different Ada compilers are allowed to choose different elaboration orders. One order might work, and another order might raise Program_Error.

In order to write portable code, you have to sprinkle elaboration control pragmas all over the place (mostly pragma Elaborate_All).

And doing that by hand is a super-human task. That's a language problem -- you can't blame any particular Ada compiler.

Note that GNAT has a switch that will tell you where to put pragma Elaborate_All.

Again, I suggest reading the elaboration section of the GNAT docs -- it explains all this stuff in great detail.

[…]

On pure functions

From: Martin Dowie <martin@thedowies.com>
Date: Thu, 12 Jan 2012 02:44:56 -0800
Subject: Pure function aspect?...
Newsgroups: comp.lang.ada

Now Ada has bitten the bullet and allowed "in out" mode parameters in functions, is it time to allow users to contract the other extreme and allow a function to be declared Pure (no state changes, not even hidden ones)? e.g.

```
package P is
   type T is tagged private;
   function Pure_F   (Self : T)
      return Integer
      with Pure;
   function Impure_F (Self : T)
      return Integer;
private
   type T is tagged record
      I : Integer := 0;
   end record;
end P;
```

Functions with a Pure contract would be allowed to call other functions with pure contracts, read values/parameters but promise to change nothing (not even via 'tricks' a la random number generator!!).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 12 Jan 2012 18:00:05 -0600
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

We've argued this for a long time within the ARG, and we haven't been able to get a real consensus. This discussion goes back to Ada 79 (which is even before I got involved in Ada) -- early drafts of Ada had both functions (which were what we now call pure) and value-returning procedures (which were not). The concern was that there was not a clear line between them, and moreover there are many sorts of functions that are

"logically" pure but still do change things (the "memo function" being the banner carrier for that idea).

Personally, I would be happy to have strong checks on such functions, and I don't much care about what gets left out (it's not that important, and any such functions are not task-safe anyway, so it is already a good idea to avoid them in Ada code). But not everyone agrees. We just had another version of this discussion in Denver; we ended up adopting an undetectable bounded error for this case (in the case of assertions, including preconditions, et. al.).

I had tried an alternative approach for Ada 2012, by suggesting the addition of checked global in/out contracts to subprograms. Eventually, this was dropped from Ada 2012 as being insufficiently mature. My understanding is that AdaCore is experimenting with a version of it in their formal methods research, so it isn't necessarily gone forever (which is good, considering the amount of time I put in on it). You can see the last proposal at

http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05s/ai05-0186-1.txt.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 13 Jan 2012 09:45:46 +0100
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

I think that the problem may be resolved by considering the contexts where the function is pure. There is no absolutely pure functions, any one is impure because it returns something, pumps stack etc. The result is taken out of consideration, so are local variables etc. There should be some syntax for specifying what is not touched at the given level of "purity" and what is.

A related issue is a requirement that functions impure only in their results and locals were evaluated at compile time when arguments are statically known. [Needed for handling dimensioned values and similar static checks, e.g. matrix dimensions checks etc]

From: Stefan.Lucks <Stefan.Lucks@uni-weimar.de>
Date: Fri, 13 Jan 2012 11:48:55 +0100
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

> I had tried an alternative approach for Ada 2012, by suggesting the addition of checked global in/out contracts to subprograms. Eventually, this was dropped from Ada 2012 as being insufficiently mature.

Very regrettable!

But reading the AI makes one understand why this is so complicated. There are class-wide-operations. If any pure function (or rather, any side-effect-free function or procedure) is going to use them, these need their own aspect annotations. These are the descendants from Ada.Finalization.*. There are instances of generic subprograms. Even if the generic subprogram is side-effect-free by itself, the side-effect-freeness of the instance is likely to depend on the side-effect-freeness of the generic parameters…

However, another reason, why the AI became so complex, seems to be the attempt to rather precisely specify side-effects, instead of providing just the ability to declare "no side effects". Now it is too late, but a simplified approach, allowing only "with Global in out => null;" with the option to extend this later would have been acceptable for Ada 2012. :-/

BTW, why do you write "with Global in out => (null);" with brackets?

From: Martin Dowie <martin@thedowies.com>
Date: Fri, 13 Jan 2012 03:01:35 -0800
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

Care to mock up an example of the levels you had in mind?

Is it something like (and stealing Randy's "Global"):

```
function F1 (P1 : T1; P2 : T2)
  return Boolean
  with Pure => (Global, P1);  -- promise
   --  to not change globals or parameter
   -- P1, might tamper with P2


function F1 (P1 : T1; P2 : T2)
  return Boolean
  with Pure => (all);  -- promise to not
    -- change anything (other than
    -- subprogram local objects)


function F1 (P1 : in out T1)
  return Boolean
  with Pure => (Global);  -- promise to
    -- not change anything other than
    -- subprogram local objects or
    -- parameters
```

A function with no pure aspect might be equivalent to:

```
function F1 (P1 : T1; P2 : T2)
  return Boolean
  with Pure => (null);  -- no promise to
    -- not change anything local, global
    -- or parameter
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 13 Jan 2012 18:12:58 +0100
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

Global = Standard (the topmost Ada package is Standard).

A body is impure in some nested context and pure outside it. So the meaning is not pure-in-A, but rather pure-outside-A, while impure inside A. E.g.

```
with Pure => (out Foo)
-- Pure outside the specification of Foo,
-- can change parameters

with Pure => (out body Foo)
-- Pure outside the body of Foo,
-- cannot change parameters

with Pure => (in Standard)
-- Pure everywhere = static, constant

type Func is not null
   access function (X : Float)
      return Float
      with Pure => …;
function Integrate (F : Func)
   return Float
   with Pure => (out Integrate and
                 out Func);
   -- Pure outside the specification of
   -- Integrate everywhere
   -- Func is pure as well
```

> A function with no pure aspect might be equivalent to:
>
> function F1 (P1 : T1; P2 : T2) return Boolean
>
> with Pure => (null); -- no promise to not change anything local, global or parameter

```
function F1 ...
  with Pure => (out Standard)
  -- impure, out Standard = empty set =
  -- null
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 13 Jan 2012 18:00:46 -0600
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

> However, another reason, why the AI became so complex, seems to be the attempt to rather precisely specify side-effects, instead of providing just the ability to declare "no side effects".

[…]

There has been strong opposition to a "checked" pure function declaration (I had floated that first, and it went nowhere), and global in out => null is essentially the same thing. The reason is the "memo function" idea (more sensibly, a function that logs its calls but is otherwise pure) is not allowed. I was trying to do something broader that could get more support.

And just declaring "no side-effects" without checking it is actively harmful in my opinion because, perversely, it makes a program less safe. That's because the compiler is going to take advantage of this declaration to remove calls (especially in contracts and assertions), and if the call in fact has side-effects, doing that is not safe (and can easily lead to bugs or even erroneous execution). (And if you aren't going to let the compiler take advantage of this knowledge, declaring it is pointless.)

I have some lengthy examples of this problem that are too long to present here (and at least get any other work done), but trust me, it is very real.

The lack in global in out isn't a deal breaker for Ada 2012, simply because it is also missing exception contracts. And we need (at least) both of those to have enough completeness to really reason formally about Ada programs.

Hopefully, future versions of Ada will take up both of these again.

From: Martin Dowie <martin@thedowies.com>
Date: Mon, 16 Jan 2012 02:48:28 -0800
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

[…]

You mean people want a method of saying "these state changing calls are unimportant"?

```
with Logger;   -- Perhaps allowing
-- 'limited with' if only needed in
-- the aspect?
package … is
  function Foo (P1 : Integer)
    return Integer
    with Pure => all or
        (Logger.Report (S : String) and
         Logger.Report (S : String;
                        I : Integer));
…
```

I think you'd need to enumerate all 'ignorable' state-changing calls.

Or perhaps a short-hand:

```
with Logger;   -- Perhaps allowing
-- 'limited with' if only needed in
-- the aspect?
package … is
  function Foo (P1 : Integer)
    return Integer
    with Pure => all or (package
                         Logger);
…
```

> […] The lack in global in out isn't a deal breaker for Ada 2012, simply because it is also missing exception contracts.

Something like:

```
function Foo (P1 : Integer)
  return Integer or raise Program_Error
                      or My_Defined_Error;
```

Getting a bit long winded but I guess you just need all that sort of information.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 18 Jan 2012 19:17:45 -0600
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

> […] You mean people want a method of saying "these state changing calls are unimportant"?

Something on that line. The global in/out proposal handling that in terms of allowing the specification of what state is modified. If that state doesn't conflict with other routines, then optimization is still possible.

But one of the important points is that side-effects really ought to prevent optimization (and assumptions) unless the compiler (or proof tool) can actually prove they don't matter. A call that is being logged, for instance, probably shouldn't be eliminated even if the results are the same (because you couldn't easily relate the actual calls to the source code).

One can argue the other side of this, too, and it's obvious there are no easy answers. In which case I prefer to stay as close to the "canonical" semantics as possible.

> […] Getting a bit long winded but I guess you just need all that sort of information.

Well, clearly it would be an aspect (like the other contracts), and each exception ought to be able to have an optional postcondition as well (which defines when the exception might be raised). So more like:

```
function Foo (P1 : Integer)
  return Integer
  when Pre => …
     Post => …
     Raises => Program_Error,
               Storage_Error,
               My_Defined_Error
                 when P1 not in Even,
     Global in => null,
     Global out => null;
```

The exception contract would require that the compiler prove that Constraint_Error and Tasking_Error are not propagated. (The same would be true for Storage_Error, but that would be impossible for the vast majority of compilers). I'm also proposing a way to name a group of exceptions, so you wouldn't have to write huge sets repeatedly. (All of the exceptions in package IO_Exceptions could be named "IO_Exceptions" so it wouldn't be necessary to write about them individually.)

I of course have no idea how far any of this will get, it won't be taken up for a while and won't be standardized for years, and in any case, it is all optional. No one is going to be required to have complete contracts (I suspect that will be going too far).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 19 Jan 2012 11:09:28 +0100
Subject: Re: Pure function aspect?...
Newsgroups: comp.lang.ada

> […]function Foo (P1 : Integer) return Integer

```
      when Pre => …
           Post => …
           Raises => Program_Error,
                     Storage_Error,
                     My_Defined_Error
                    when P1 not in Even,
```

The problem here is that the exception contracts do not obey the law of excluded middle. I.e. raise A or not raise A is not necessarily true. The syntax should distinguish promises (not to raise A) with obligations (to raise A). In the above you have a negated promise not raise Program_Error, i.e. Foo *may* raise Program_Error, but it is not required to do so. On the contrary, My_Defined_Error is an obligation, Foo is *required* to raise it when the condition is not met.

This is "intuitionistic logic," a bit complicated thing, but necessary here. The simplest way to handle it would be to have:

```
function Foo …
  <a-nice-name-for-possible-exceptions>
      => X,
  <necessary-exceptions> => Y
```

Y is a subset of X. Foo never raises anything from not X.

Inference of contracts of possible exceptions is simple. A conservative estimation is a union of all exception sets of called subprograms for which no handler is present.

For necessary exceptions it is in general impossible to do (equivalent to halting). So it looks more appropriate for SPARK than for Ada.

> Global in => null,
  Global out => null;

> The exception contract would require that the compiler prove that Constraint_Error and Tasking_Error are not propagated. (The same would be true for Storage_Error, but that would be impossible for the vast majority of compilers).

Storage_Error should be a conditional:

```
may_raise Storage_Error when
  Standard_Pool'Free_Space < 1024;
```

Actually any exception should be. E.g.

```
procedure Sort (X : in out Data;
                F : Order_Func)
  may_raise Program_Error when
    Order_Func may_raise
      Program_Error;
    -- Sort does not raise Program_Error
    -- by itself
```

> I'm also proposing a way to name a group of exceptions, so you wouldn't have to write huge sets repeatedly. (All of the exceptions in package IO_Exceptions could be named "IO_Exceptions" so it wouldn't be necessary to write about them individually.)

It is time to make exception a proper discrete type with another type for sets of exceptions. It would be nice to have a tree order of exceptions, i.e. extensible sets of exceptions too.

> […] No one is going to be required to have complete contracts (I suspect that will be going too far).

No need to have complete contracts because there is a gray area between 'may raise' and 'must raise'.

On expression functions vs. body of functions

From: Martin Dowie <martin@thedowies.com>
Date: Fri, 16 Dec 2011 04:25:09 -0800
Subject: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

Are there any good arguments for *not* replacing all simple, single line functions that don't [directly] access package body state information with expression functions?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 16 Dec 2011 14:24:15 +0100
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

1. Readability
2. Proper encapsulation (to have interface and implementation separated)
3. Re-use (the same function must be refactored)
4. Maintainability (because of 1..3)
5. Safety (proper bodies are defined on the context where they have no access to the caller's context, otherwise than through parameters)
6. Deployment (proper bodies can be put into a library, have versions etc)
7. It is not Ada

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 16 Dec 2011 19:03:46 -0600
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

> […]

Umm, an expression function is just another (shorter) way to write a function body, so it hard to imagine that there is any difference. Using them to replace a function body *in place* is completely harmless.

I suppose you are talking about the use of expression functions directly in a specification (without an explicit body), which is a totally separate issue. There, I tend to agree with you in the sense that they ought to be used in moderation. The intent was to use them for things like accessors for private components where there is no real value to the separate body.

For those sorts of uses, the issues you talk about above don't arise. (How could you "refactor" Obj.Field??). Moreover, everything in the private part is part of the implementation anyway; you're still enforcing that separation so long as the expression functions are in the private part.

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 16 Dec 2011 10:03:07 -0800
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

If you're talking about a function that is declared in the visible part of a package specification, and talking about replacing the declaration with an expression function, then it would usually be a bad idea to replace it. The visible part should, ideally, express what the package is intended to accomplish, at a conceptual level. So unless the expression is, itself, part of the concept (a rare case), it's an implementation detail that should not be present in the visible part of a package. A good test here, I think, is: "Is it possible that at some later point, I may change the expression returned by an expression function because I've added new features to the package or because I've changed the implementation to improve its efficiency or fix problems?" If the answer is yes, then the expression is probably an implementation detail and not part of the package's "concept", and the function shouldn't be an expression function. I think this is approximately what Dmitry means by "proper encapsulation".

Offhand, I don't see any problem with replacing simple function *bodies* with expression functions. But I haven't studied the new rules carefully, so I couldn't tell you if there are some "gotchas" in the language rules that would cause the semantics or the impact on other language rules to be different.

In most cases, it's probably OK to replace a function specification in the *private* part of a package with an expression function if the body is simple. But the encapsulation argument could still apply if the function is intended to be usable by child packages.

From: Georg Bauhaus
Date: 17 Dec 2011 12:26:21
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

A pragma Pure is, I think, helpful when writing expression functions in particular if it forces thinking about the kind of functions one is writing: expression functions that have effects other than computing the result value seem out of (some) style.

From: Martin Dowie <martin@thedowies.com>
Date: Fri, 16 Dec 2011 14:36:12 -0600
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

The most common usage I can image is functions used in Pre/Post conditions:

```
package Foos is
  type Foo is tagged private;
  …
  procedure Bar (Self : in out Foo)
    with Pre => Is_Valid (Self);
  function Is_Valid (Self : Foo)
    return Boolean;
  …
private
  …
  function Is_Valid (Self : Foo)
    return Boolean is (<validate_Self>);
  …
end Foos;
```

There isn't really an expectation that Is_Valid would be of use to a user of package Foos.

But I guess the same idiom could be used for other 'simple' functions…public declaration, private expression function implementation, leaving the package body of more interesting stuff.

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 16 Dec 2011 13:34:51 -0800
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

I don't see a problem (from a proper encapsulation standpoint) with putting the expression function in the private part.

Like I said before, though, I haven't studied the new rules enough to know whether there could be any other problems caused by the language rules. Offhand, it seems possible that if the expression involves a call to a function in another package, there could be cases where moving the definition from the body to the specification could fail because a different elaboration order is required.

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 16 Dec 2011 15:08:13 -0800
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

On thinking about it further: A good reason *not* to put an expression function in the private part of a specification, if the function is an "implementation detail", is that if the details of the implementation are mostly in the package body (in procedures or in more complex functions), then moving some of it arbitrarily to the private part, which is separate from the package body and is often in a different source file, can impair readability and maintainability. When a programmer is trying to maintain a package, either by adding new features, improving performance, or fixing problems, it's helpful for related pieces of information about the package's implementation to be near each other. Otherwise, it's too easy to miss something. And moving one implementation detail to a package's private part while other details are still in the body increases the chance of missing something.

So yes, I think there is a good argument for *not* mechanically moving all simple functions to the package specification as expression functions, even into the private part. The programmer needs to think about readability/maintainability issues such as I've described.

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 16 Dec 2011 14:52:01 -0800
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

> […] Are there any good arguments for expression functions?

Quote from AI05-0177-1 (http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05s/ai05-0177-1.txt?rev=1.13):

With the advent of pre and postconditions (see AI05-0145-1), and conditional expressions (see AI05-0147-1), expressions in specifications are going to grow much larger and become more complicated. It is important that parts of such expressions can be abstracted.

Abstraction of expressions is usually done by introducing functions. However, this puts the expression in the body, rather than the specification. This has several negative side-effects:

- Hiding of the expression from the compiler and other tools that primarily process specifications;
- Requiring the body to exist in order to do static analysis of the pre/post conditions (meaning static analysis cannot be performed early in the development of a system or on the use of skelton [sic] placeholder packages).
- Introduction of otherwise unnecessary bodies and/or otherwise unnecessary inline body dependencies (increasing system build times).

Apparently the ARG thought this was a good argument. I'm not endorsing it personally -- I don't have a particular opinion -- but it appears sensible on its face.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 16 Dec 2011 19:21:45 -0600
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

One of the things we learned when thinking about Preconditions and the like is that there is such a thing as too much encapsulation. The entire point of preconditions is that they be understandable to the client, so the client (caller) knows what they must ensure before calling the routine.

You could write all of your preconditions like:

```
procedure Do_It (A, B : in out Integer)
   with Pre => Do_It_Precondition (A, B);
```

but no one would have any idea what the precondition is. After all, an important part of the point of preconditions is to change what currently is specified in English in the comments in a more formal way that can be checked (statically and dynamically).

So it is best to use as little encapsulation as possible in preconditions and the like, breaking them down to the primitive operations of the type(s) involved.

But preconditions can get very long, and the need to simplify them for readability surely exists. Thus expression functions provide a middle ground. They're also handy for simple accessor functions (which can give the effect of read-only components of a private type).

There also is one more fact: an expression function can almost always be inlined (beware of recursion, though), and that can be done automatically when it makes sense (no need for pragmas or body dependencies -- inlining is something the compiler should be deciding upon anyway, it is stupid for users to have to declare something that should always be done if it makes sense -- and not be done when it doesn't make sense).

From: Georg Bauhaus
Date: 17 Dec 2011 12:45:38
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

I'm guessing that Dmitry will suggest

```
procedure Do_It (A, B: Int_Sats_Pre)
   with Pre => True;
```

will be safer and will convey the idea of the precondition better: it is in the type system.

Whether this approach is feasible in general I don't know.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 17 Dec 2011 14:11:35 +0100
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

Not really. The key question is whether Do_It_Precondition is statically checkable. Note also that it is not always possible to break a [true] precondition into a set of *independent* subtype constraints. As an example consider:

```
function "+" (Left, Right : Dimensioned)
   return Dimensioned;
```

The precondition here (IFF measurement units have to be checked statically) is that Left and Right have the same unit.

If the measurement units cannot be checked statically THEN the precondition is "true" and the contract of "+" includes Unit_Error.

Argument against Do_It_Precondition is same as against a formula:

declarations should include minimum executable code. The language of declarations (types algebra operations) must be clearly, visibly separated from the object (executable) language. All cases when executable code slips

```
procedure Foo
   (Default : Integer := Get);
   -- What is the default here? When the
   -- default is read from standard input?
```

Pure and static expressions are OK because the context is irrelevant.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 19 Dec 2011 17:34:42 -0600
Subject: Re: Ada2012 : When to use expression functions rather than function bodies?
Newsgroups: comp.lang.ada

[…]

Right. We had this (sub)discussion in the ARG. It seemed better to extend subtype constraints for parameters rather than the heavier mechanism of preconditions. But the counter argument is that a constraint can act only on a single parameter, while a precondition might involve multiple parameters.

Dmitry shows a good example.

BTW, we recently added a rule stating that it is a bounded error to call a function from a contract (precondition, predicate, etc.) that has a side effect that changes the value of some other contract of the same evaluation. The latter part is a sop to the people who insist that we have to support "benign" side-effects (such as "memo functions"). (IMHO, there are no benign side-effects, but there are strong opinions to the contrary out there.)

The effect is that the vast majority of contracts will be "pure" expressions, so Dmitry will be happier. (The state-of-the-art will not allow static checking of these things today in general, but as the technology improves that should become possible without having to rewrite your code).

On task abortion

From: tonyg <tonythegair@gmail.com>
Date: Wed, 25 Jan 2012 02:11:55 -0800
Subject: task abortion
Newsgroups: comp.lang.ada

I want to be able to abort a task if it has not responded for a while, and then start a new one in its place.

I have a pointer going to the concerned task.

Each task has a 'stop accept' in its rendezvous which forces it out of its loop so that it ends, but if the task has frozen for some reason then I want to abort.

I would be interested how to do this and folks strategies for keeping on top of tasks.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 25 Jan 2012 11:43:14 +0100
Subject: Re: task abortion
Newsgroups: comp.lang.ada

[…]

What are you going to achieve by that?

Consider the scenarios:

1. The task crashed, it is already terminated then.

2. The task is looping somewhere:

2.a. What about the resources it owns? Local resources are usually freed when the task is aborted. But if you have some globally allocated memory, semaphores etc, they all get lost unless freed by some local controlled objects, "holders" releasing the resources upon finalization, as they leave the scope upon task abort. Now:

2.b. What if the task is frozen in an abort-deferred thing? Finalization is such a thing. Abort-deferred stuff cannot be aborted. Note that system I/O is most likely non-abortable. I.e. you would not be able to abort Ada.Text_IO.Get_Line or recv on a socket etc anyway.

All in one, aborting tasks doing complex stuff is not likely to work in a reasonable way. Aborting tasks doing simple stuff is not needed, such tasks impose no problems anyway. Ergo, it is not likely you would need to abort any task.

If you decided for aborting, you would have to redesign almost everything and in an extremely careful way in order to make tasks logically abortable.

That means to make continuation possible and meaningful after aborting some tasks. Now a guess: making the tasks right and thus preventing a need aborting them, would probably be far less efforts…

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Thu, 26 Jan 2012 08:47:16 -0800
Subject: Re: task abortion
Newsgroups: comp.lang.ada

[…]

Just in case your task disappear which I do not expect it will, you can use package Ada.Task_Termination to query the reason (Normal, Abnormal, Unhandled_Exception) it dies.

Freezing/pausing a task

From: Pablo Rego <pvrego@gmail.com>
Date: Thu, 17 Nov 2011 07:33:24 -0800
Subject: Freezing a task
Newsgroups: comp.lang.ada

Is is possible to freeze a task?

I mean, if I have a task

```
task body My_Task is
begin
  accept Start;
  loop
    Put ("1");
    Put ("2");
    Put ("3");
    …
    Put ("n");
  end loop;
end My_Task;
```

is there a way that I can "freeze" the task in its current state?

If, for instance, the execution finished executing Put ("2");, how can I freeze it and later I can turn it to continue? I want to freeze from outside the task, and also from outside, order it to continue.

I could sure implement, if I had the spec like

```
type State_Type is (RUN, FROZEN);

task type My_Task (State : State_Type)
is
   entry Start;
end My_Task;
--
```

the body:

```
task body My_Task is
begin
  accept Start;
  loop
    Put ("1");
    Put ("2");
    Put ("3");
    …
    Put ("n");

    loop
     if State = RUN then exit; end if;
    end loop;
  end loop;
end My_Task;
```

but it would not be the case because I had to wait for the nth Put instruction line (i.e., the task would not be actually frozen, because the inside loop would be running).

And T.E.D. from stackoverflow suggested me something that I could infer as

```
task type My_Task (Start : Start_Type)
  is
   entry Start;
   entry Run;
end My_Task
--
task body My_Task is
begin
  accept Start;
  loop
    Put ("1");
    Put ("2");
    Put ("3");
    …
    Put ("n");

    if State = FROZEN then
      accept Run;
      State := RUN;
    end if;
  end loop;
end My_Task;
```

Is there a more elegant way to do this?
[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 17 Nov 2011 17:00:40 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

> Is is possible to freeze a task?

Yes, by

1. an entry call to another task or a protected object;

2. executing a select or accept statement.

From: Simon Wright <simon@pushface.org>
Date: Thu, 17 Nov 2011 16:00:17 +0000
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

Programmatically, no. Your choices for inter-task comms are (a) the rendezvous, (b) polling some shared resource (e.g. a protected object), (c) something using sockets.

Or you could unilaterally abort the task, which would be the end of it.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 17 Nov 2011 17:22:57 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

If using select statement for accepting entry calls then:

```
   Paused : Boolean := False;
begin
   loop
      if Paused then
         accept Release;
         Paused := False;
      else
         select
            accept Pause;
            Paused := True;
         else
            … -- Do stuff
         end select;
      end if;
   end loop;
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 17 Nov 2011 17:53:42 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

P.S. there is a deadlock in above if improperly used from outside. It is simple to fix.

It is worth to note that one of the rationales behind protected objects introduced in Ada 95, was to ease things like above. The same done with a protected object would be simpler, safer and reusable (within the task) too.

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Subject: Re: Freezing a task
Date: Thu, 17 Nov 2011 17:53:46 +0100
Newsgroups: comp.lang.ada

[…]

What do you really want? Why do you want a task to freeze?

If you just want to freeze a task to ensure another more important task isn't slowed down (and unfreeze when the other task is done) a MUCH more elegant way to do this is to use task priorities.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 17 Nov 2011 18:08:17 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

Maybe, however it would not work as the given example does I/O.

Unless you have access to very high level OS priorities (which you normally don't) you won't be able to stop the system services, drivers and interrupt routines doing I/O on behalf of the task. Not that you really wanted it anyway…

IMO, in probably 90% all cases playing with priorities is a bad idea. The rest 10% require a very careful upfront analysis.

From: Adam Beneschan <adam@irvine.com>
Date: Thu, 17 Nov 2011 09:13:53 -0800
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

Aside from the other suggestions, you might want to look into Ada.Synchronous_Task_Control and Ada.Asynchronous_Task_Control. I'm not really clear on what you're trying to accomplish, so it's hard for me to say whether those are appropriate solutions for you. I think that the other methods that have been suggested--an entry call on another task or on a protected object, or an ACCEPT statement--would be preferable if they get the job done.

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Thu, 17 Nov 2011 10:01:11 -0800
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

> Aside from the other suggestions, you might want to look into Ada.Synchronous_Task_Control and Ada.Asynchronous_Task_Control.

Synch might do the job, asynch is (for GNAT) only implemented on bare boards.

From: Pablo Rego <pvrego@gmail.com>
Date: Thu, 17 Nov 2011 17:34:48 -0800
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

Well, answering also Jeff, Adam and Stefan, there are some bots which are shared by some tasks, and each task controls just one subsystem reciprocally (i.e. this one is controlled at one timeslot by just one task), but other tasks can "steal" the thread, so they can assume the control over that ones. But each task has also a different heuristics, so when it assumes a bot, it behaves different, but I want them to be restored to any other previous controllers.

Due to this I needed to "pause" that task. I guess Dmitry answer could be used very well for this (but also Simon's).

From: Jeffrey Carter <jrcarter@acm.org>
Date: Thu, 17 Nov 2011 23:04:54 -0700
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

> […] and if I would want to add an abort entry? What should I do?

Depends on when you want to be able to stop the task. I'd guess

```
select
   accept Pause;
   Paused := True;
or
   accept Stop;
   exit;
else
   -- Do stuff.
end select;
```

"abort" is a reserved word, so it can't be the name of the entry. A protected object seems cleaner, though:

```
protected Control is
   procedure Process;
   -- Instruct the task to start processing,
   -- or to resume processing after
   -- being paused.
   procedure Stop;
   -- Tell the task to terminate.
   procedure Pause;
   -- Tell the task to pause processing.
   entry Get (Stop : out Boolean);
   -- Used by the task to wait until
   -- it should do something.
   -- Stop will be True if the task should
   -- terminate; False if it should
   -- do stuff.
private -- Control
   …
end Control;

task body T is
   Stop : Boolean;
begin -- T
   loop
      Control.Get (Stop => Stop);
      exit when Stop;
      -- Do stuff.
   end loop;
```

```
end T;
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Fri, 18 Nov 2011 09:47:58 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

Note that the above has a deadlock when you call to Pause or Release twice.

It is easy to correct, but a protected object solution would be more robust. See the Jeffrey's response, however, I would use an exception rather than output parameter:

```
Pending_Abort : exception;
protected type Control is
   procedure Process;
   procedure Pause;
   procedure Stop;
   entry Get;
private
   Paused : Boolean := False;
   Exiting : Boolean := False;
end Control;

protected body Control is
   procedure Process is
   begin
      Paused := False;
   end Process;
   procedure Pause is
   begin
      Paused := True;
   end Pause;
   procedure Stop is
      Exiting := True;
   end Stop;
   entry Get when not Paused
                     or Exiting is
   begin
      if Exiting then
        raise Pending_Abort;
      end if;
   end Get;
private
   Paused : Boolean := False;
   Exiting : Boolean := False;
end Control;

task body T is
begin
   loop
      Control_Instance.Get;
      … -- Do stuff
      Control_Instance.Get;
      … -- Do other stuff
      Control_Instance.Get;
      … -- Do yet another stuff
   end loop;
exception
   when Pending_Abort =>
      null;
end T;
```

You don't need Start entry of the task. Just make Paused initially true.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Fri, 18 Nov 2011 09:56:24 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

> Well, answering also Jeff, Adam and Stefan, there are some bots which are shared by some tasks, and each task controls just one subsystem reciprocally (i.e. this one is controlled at one timeslot by just one task), but other tasks can "steal" the thread, so they can assume the control over that ones.

This is what rendezvous are for. If the task A and the task B must cooperatively do some stuff (= control a bot), they just engage a rendezvous and perform that stuff upon the rendezvous. That would effectively block one of them.

P.S. I don't understand this design. Why do not you have a pool of tasks and assign a free one for each operation to be performed on the bot? Once completed the operation, the task returns to the pool.

From: Simon Wright
<simon@pushface.org>
Date: Fri, 18 Nov 2011 10:05:10 +0000
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

> however, I would use an exception rather than output parameter

I like that. Will give it strong consideration for a design update (of course, it wouldn't do for a SPARK implementation; personally I'd like to see SPARK allow provably-handled exceptions (though I suspect the problem is with "provably")).

From: Georg Bauhaus
Date: Fri, 18 Nov 2011 12:41:41 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

In case I want an interrupt to stop (and release) the task, will an exception, such as Pending_Abort from the example, work? Because, IIUC, when the protected procedure handling the interrupt raises Pending_Abort, then this has no effect (as the procedure is a handler, LRM C.3(7)). Is this correct?

Maybe the handler procedure might just adjust the status of the object so as to take note of the interrupt.

When the task then calls an entry such as Get from the example, and the entry raises an exception, it will have an effect. But then, this will interrupt the task only later…

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Fri, 18 Nov 2011 14:42:06 +0100
Subject: Re: Freezing a task
Newsgroups: comp.lang.ada

[…]

> In case I want an interrupt to stop (and release) the task, will an exception, such as Pending_Abort from the example, work?

Since the entry point Get is to be called on the context of a task, Pending_Abort will never propagate on the context of an interrupt.

> When the task then calls an entry such as Get from the example, and the entry raises an exception, it will have an effect. But then, this will interrupt the task only later…

This is exactly the implementation. BTW this is also the case for the terminate alternative, which is accepted when the task is ready to accept it.

On the visibility of entries of a protected object

From: Jacob Sparre Andersen
<sparre@nbi.dk>
Date: Thu, 09 Feb 2012 12:56:06 +0100
Subject: Publishing selected entries in a protected object?
Newsgroups: comp.lang.ada

I have a package containing a protected object, and I would like to publish _one_ of this object's entries in the specification of the package.

Since I want to be able to use the entry in a select statement, I can't just encapsulate it in a procedure and make that available in the package specification.

And if I move the declaration of the protected object to the package specification, but move the other entries to the private part of the protected object, I can't access the other entries from the private part of my package.

Is there a pattern I can use?

Thanks in advance,

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Thu, 9 Feb 2012 14:42:19 +0100
Subject: Re: Publishing selected entries in a protected object?
Newsgroups: comp.lang.ada

[…]

Not exactly, but…

```
type I is protected interface;
procedure A (X : in out I) is abstract;

protected type Foo is new I with
   overriding entry A; -- "Public entry"
```

```
   entry B; -- "Private entry"
end Foo;
```

Now, if only the interface I is visible, then that would be close to what you wanted.

P.S. Alas, but the following is illegal:

```
 [protected] type Foo is new I
   with private;
private
   protected type Foo is new I with …
   end Foo;
```

From: Georg Bauhaus
Date: Thu, 09 Feb 2012 14:59:51 +0100
Subject: Re: Publishing selected entries in a protected object?
Newsgroups: comp.lang.ada

[…]

Using synchronized interfaces should work?

```
package Semi is
   type Half is synchronized interface;
   procedure Grab (PO : in out Half)
     is abstract;
   function MakePO return Half'Class;
private
   protected type Full is new Half with
     overriding entry Grab;
     entry Foo;
   end Full;
end Semi;

function MakePO return Half'Class is
begin
  return Result : Full;
end MakePO;
```

From: Jeffrey Carter <jrcarter@acm.org>
Date: Thu, 09 Feb 2012 12:29:59 -0700
Subject: Re: Publishing selected entries in a protected object?
Newsgroups: comp.lang.ada

[…]

```
package P is
   protected Public is
      entry E;
   end Public;
end P;

package body P is
   protected Hidden is
      entry E;
      entry Internal;
   private -- Hidden
      …
   end Hidden;

   protected body Public is
      entry E when True is
        -- null;
      begin -- E
         requeue Hidden.E [with abort];
      end E;
   end Public;
   …
end P;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 9 Feb 2012 19:29:39 -0600
Subject: Re: Publishing selected entries in a protected object?
Newsgroups: comp.lang.ada

> Using synchronized interfaces should work?

It will often work, but there are some problems. We couldn't get the design of the Queue containers to work out properly, and we ended up with the horrifically ugly "Implementation" package.

I think that can be fixed, but it will require some new capabilities for protected types, and those may cause other problems. Oh well, Ada has to have some problems or the ARG wouldn't need an editor, and then I'd have to find some real work. ;-)

On Vectors and the array type

From: Ada BRL <ada.brl.2011@gmail.com>
Date: Sat, 21 Jan 2012 10:19:06 -0800
Subject: Efficiency and overhead: Ada.Containers.Vectors.vector versus array type
Newsgroups: comp.lang.ada

Hello everyone!

I need some hints about Ada.Containers.Vectors.vector efficiency.

I'm developing a multi-threaded real time application in Ada.

I need a "collection" of objects (every object has a lot of records like task objects, GNAT.Sockets and so on…);

This collection is accessed several times during the execution.

In the meantime I don't have to insert and delete any items during the execution, I just need to instantiate N object when the application starts and then the number of objects will remain the same throughout the execution.

Since I know how many object will be inside the collection I thought to use, as the collection I need, the "array standard type".

In the meanwhile the Ada.Containers.Vectors.vector object is far more practical, simple and has even more interesting and useful functions like iterators and so on.

For this reason, since I don't have any experience in Ada, I ask you all what can you suggest me to use.

If there is no such difference in efficiency/overhead between array types and Ada.Containers.Vectors.vector I'll definitely use the latter.

From: Simon Wright <simon@pushface.org>
Date: Sat, 21 Jan 2012 18:40:52 +0000
Subject: Re: Efficiency and overhead: Ada.Containers.Vectors.vector versus array type
Newsgroups: comp.lang.ada

[…]

It's bound to take longer to, for example, iterate over a Vector than to loop over an array. The question is, how much longer? And is that acceptable for your proposed usage?

I'd have thought the actual operations you want to do on your objects would be independent of the storage mechanism, so maybe you could use the Vectors first, do some measurements, and see if that will be adequate.

I'd start by something like creating an Indefinite_Vector of Strings, populating it with say 1000 strings, and seeing how long it takes to iterate over it. I'd expect it to be sub-microsecond per element, on a modern processor (but I could be wrong!)

One thing to be wary of: an object with an embedded task will be limited, which means you can't hold it directly in any of the standard Containers; you'd have to hold access-to-object (or access-to-object'Class if you have tagged types). This means you'll need to do storage management yourself, perhaps using Ada.Finalization.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Sat, 21 Jan 2012 12:11:31 -0700
Subject: Re: Efficiency and overhead: Ada.Containers.Vectors.vector versus array type
Newsgroups: comp.lang.ada

[…]

This sounds perfect for an array. What is called a "vector" in the standard container library is an unbounded array. One uses an unbounded data structure when one doesn't know how large the structure will be until run time in such a way that one cannot declare a bounded structure; one uses a bounded structure (an array, in this case) in most other cases.

There's an additional reason you probably want an array rather than a vector: since your objects contain tasks, they are limited. You can't store limited objects in a vector, so you'll have to allocate them on the heap and store accesses to them in the vector. This introduces additional complexity to your code that would not appear when using an array. Also, since a vector is unbounded, it is also stored on the heap and accessed through an access value, making a vector involve double indirection.

Usually the syntax for dealing with an array is clearer than the equivalent using a vector (in current Ada; the next version of the standard will include changes to make them more equivalent).

On initialization of array size

From: Nasser M. Abbasi
Date: Sat, 04 Feb 2012 05:00:11 -0600
Subject: can one create an array of a size determined by a constant value of a function?
Newsgroups: comp.lang.ada

Quick question for the experts:

I was looking at C++11 standard, where it show how one can allocate an array of some size. The size is determined by a function call. This is done at compile time though where the compiler can determine the result of the function, like this:

http://en.wikipedia.org/wiki/C%2B%2B11#cite_note-2

"constexpr int get_five() {return 5;}

int some_value[get_five() + 7]; // Create an array of 12 integers. Legal C++11

This allows the compiler to understand, and verify, that get_five is a compile-time constant."

constexpr is new and was added in c++11

Is it possible to do something like this in Ada?

I am not sure now how useful or common such a feature would be actually, but was just wondering how it will look in Ada.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Sat, 04 Feb 2012 11:47:29 -0700
Subject: Re: can one create an array of a size determined by a constant value of a function?
Newsgroups: comp.lang.ada

[…]

With Ada, you can declare an array with a size not known at compile time:

```
function Get return Positive is
   -- null;
begin -- Get
   return Integer
      (Ada.Calendar.Seconds
                (Ada.Calendar.Clock) );
end Get;

S : String (1 .. Get);
```

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Sat, 04 Feb 2012 13:36:35 -0500
Subject: Re: can one create an array of a size determined by a constant value of a function?
Newsgroups: comp.lang.ada

[…]

You can't declare that a function will return a compile-time-known value. But if the goal is efficiency, you don't need to -- the compiler can figure it out. For example, compile the program below with "gcc -O2", and look at the generated assembly.

You'll see that Get_Five has vanished, and the array is allocated just as if you had said "1..12" -- the expression "Get_Five+7" is evaluated at compile time.

If Get_Five were separately compiled, you might need pragma Inline.

If Get_Five returned something not known at compile time, then the calculation would be done at run time, and a dynamic array would be allocated on the stack.

```
with Text_IO; use Text_IO;
procedure Main is

   function Get_Five return Integer;
   function Get_Five return Integer is
   begin
      return 5;
   end Get_Five;

   Some_Value: array (Integer range 1 ..
            Get_Five + 7) of Integer;

begin
   for X in Some_Value'Range loop
      Some_Value(X) := X;
   end loop;

   Put_Line(Some_Value(12)'Img);
end Main;
```

Block diagrams for Ada code

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Sat, 21 Jan 2012 02:27:21 -0800
Subject: Block diagrams for Ada code?
Newsgroups: comp.lang.ada

Dear all,

I need to draw a "block diagram" overview of a fairly complex Ada code and I was wondering if there was some more or less "standard" symbols to denote things like packages, tasks and protected objects.

The only thing that comes to my mind is UML, but maybe I would prefer something more Ada-specific. Any ideas?

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sat, 21 Jan 2012 12:41:58 +0200
Subject: Re: Block diagrams for Ada code?
Newsgroups: comp.lang.ada

[…]

HRT-HOOD has some graphical symbols for things like that. Google for it, or see ciclope.fi.upm.es/display/docs/HRT-HOOD.pdf.

For a design tool, check http://www.ellidiss.com/.

From: Martin Dowie <martin@thedowies.com>
Date: Sat, 21 Jan 2012 06:34:17 -0600
Subject: Re: Block diagrams for Ada code?
Newsgroups: comp.lang.ada

[…]

You could look out Ada Structure Graphs. These were big in the early 90's and did a great job of describing Ada 83 but I haven't looked them up since.

Did find this link:

http://www.threesl.com/pages/reference/diagrams/ada-structure-graph.php

From: Marc Criley
Date: Sat, 21 Jan 2012 07:31:54 -0800
Subject: Re: Block diagrams for Ada code?
Newsgroups: comp.lang.ada

[…]

SciTools' "Understand" (http://www.scitools.com/index.php) product does block and control flow diagrams, browsing, metrics, and just a whole lot. It's been my "go to" tool when I've had the need to dig into a legacy code base, including repositories of upwards of a million SLOC. They have first class support for Ada, as well as the other languages they support, and their technical support has been amongst the best in the business from my experience.

It's probably priced to high for the average one man shop user at $995, but a trial download (http://www.scitools.com/download) does include a 2-week evaluation license.

Stack information on exception raising

From: tonyg <tonythegair@gmail.com>
Date: Fri, 20 Jan 2012 07:03:44 -0800
Subject: Tracing procedural calls when an exception is raised
Newsgroups: comp.lang.ada

When using the GNAT compiler is there a way to get hold of the procedure or function stack when an exception is called?

I am using at the moment

```
Ada.Exceptions.Exception_Information
   (Error)
```

within an exception handler and I am looking for better information

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Fri, 20 Jan 2012 07:08:48 -0800
Subject: Re: Tracing procedural calls when an exception is raised
Newsgroups: comp.lang.ada

[…]

Yes, you need to use a special option at compile time and then addr2line. See

http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gnat_ugn_unw/Tracebacks-From-an-Unhandled-Exception.html

From: Simon Wright <simon@pushface.org>
Date: Fri, 20 Jan 2012 16:44:44 +0000
Subject: Re: Tracing procedural calls when an exception is raised
Newsgroups: comp.lang.ada

[…]

Are you using Mac OS X? if so, see http://goo.gl/XGpQf

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 20 Jan 2012 18:16:09 +0100
Subject: Re: Tracing procedural calls when an exception is raised
Newsgroups: comp.lang.ada

> […] using Debian :)

Then see:

```
GNAT.Traceback.Symbolic.
Symbolic_Traceback
```

It should work under Debian.

You can even let the GPS open the file and jump to the specified source line from the traceback.

RESTful web API and AWS

From: p34cekeeper <wavefront.arbiter@gmail.com>
Date: Wed, 17 Aug 2011 04:51:18 -0700
Subject: RESTful web API using AWS?
Newsgroups: comp.lang.ada

I'm currently tasked with developing a RESTful web API as a front-end for what amounts to a bunch of bog-standard RDB operations. I'm currently using PHP, which, as it stands is making me feel ill.

While investigating other (more orthogonal) technology choices (erlang, Scala, etc.), I happened to come across Ada Web Server. I suppose my question is: is AWS up to the job? Has anyone here done this before?

It seems to me like AWS' dispatcher mechanisms are ideal for this purpose, because they don't make assumptions about what you want to do with the incoming HTTP request (I'm tired of frameworks that force you into a ham-fisted, code generating MVC implementation).

Any input from experienced Ada engineers would be greatly appreciated.

From: Georg Bauhaus
Date: Wed, 17 Aug 2011 14:43:27 +0200
Subject: Re: RESTful web API using AWS?
Newsgroups: comp.lang.ada

I'd suggest asking on the low volume mailing list, too.

There is an AWS paper written by J.-P. Rosen might be of interest, if you haven't seen it yet.

It was recently linked from http://www.reddit.com/r/ada/

[…]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 17 Aug 2011 06:14:56 -0700
Subject: Re: RESTful web API using AWS?
Newsgroups: comp.lang.ada

[…]

The short answer is: yes. Feel free to ask more specific questions here or on the AWS mailing list.

Shameless_Plug : begin

AWS comes in source-only form; you have to compile and install it on your development and on all your target machines. In contrast, Debian [1] contains the precompiled package libaws2.7-dev which takes the hassle of installation and deployment away. You might want to try that for a _very_ quick installation of a complete Ada development platform including compiler and lots of goodies.

[1] http://www.debian.org

end Shameless_Plug;

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 17 Aug 2011 08:33:38 -0700
Subject: Re: RESTful web API using AWS?
Newsgroups: comp.lang.ada

[…] You'll also be interested in the packages libgnadesqlite3-1-dev, libgnadeodbc1-dev and libapq-postgresql1-dev, which contain Ada bindings to SQLite, UnixODBC and PostgreSQL respectively.

From: Alex R. Mosteo <alejandro@mosteo.com>
Date: Wed, 17 Aug 2011 18:01:02 +0200
Subject: Re: RESTful web API using AWS?
Newsgroups: comp.lang.ada

[…]

Just to add another success history for your peace of mind. I've not done web APIs with AWS, but two interfaces to quite different programs (one p2p, other mobile robotics), and AWS is of outstanding quality. Particularly if you like Ada, it's a no-brainer choice.

[…]

From: Marcelo Coraça de Freitas <marcelo.batera@gmail.com>
Date: Thu, 18 Aug 2011 06:40:02 -0700
Subject: Re: RESTful web API using AWS?
Newsgroups: comp.lang.ada

[…]

There is also APQ for MySQL and ct_lib (Sybase/SQL Server) in Debian repositories IIRC.

As using AWS for RESTful API, I have some experience in developing it myself. It's awesome not to be forced to link to a file or something of the sorts. You can organize your code as you wish.

If you want to work with JSON data, take a look at the KOW Lib project at http://framework.kow.com.br.

There is KOW_Lib.Json, which I have been using for quite some time now.

Happy coding! :)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 18 Aug 2011 07:29:32 -0700
Subject: Re: RESTful web API using AWS?
Newsgroups: comp.lang.ada

[…]

No, not in Debian unfortunately. Adrian-Ken Ruegsegger has only packaged apq and apq-postgresql so far; the other modules are only available from upstream[1] ATM.

[1] http://framework.kow.com.br/

How to read the Windows login username in Ada 95

From: Tom Moran <tmoran@acm.org>
Date: Wed, 9 Nov 2011 05:05:27 +0000
Subject: Re: Read Windows login username in Ada 95
Newsgroups: comp.lang.ada

> Is there a function in Ada 95 which returns me a String containing the logged username in an Windows application? Thanks.

No. Not all Ada applications run in a Windows environment.

In Windows, you can call function GetEnvironmentVariable and get the value of the environment variable "USERNAME".

From: Pascal Obry <pascal@obry.net>
Date: Wed, 09 Nov 2011 11:44:14 +0100
Subject: Re: Read Windows login username in Ada 95
Newsgroups: comp.lang.ada

[…]

No, for a portable solution you can use:

```
POSIX.Process_Identification.
Get_Login_Name
```

Available on Florist (GNU/Linux) and wPOSIX (Windows).

From: Pablo Rego <pvrego@gmail.com>
Date: Wed, 9 Nov 2011 12:47:57 -0800
Subject: Re: Read Windows login username in Ada 95
Newsgroups: comp.lang.ada

Well, answering myself, we could use

```
function GetUsername return String is
   function GetEnv (Variable : String)
    return Interfaces.C.Strings.chars_ptr;
   pragma Import (C, GetEnv, "getenv");
```

```
   Command : constant String :=
                              "USERNAME";
   Answer_Ptr : constant
        Interfaces.C.Strings.chars_ptr :=
                        GetEnv (Command);
   Answer : constant String :=
        Interfaces.C.Strings.Value
                         (Answer_Ptr);
begin
   return Answer;
end GetUsername;
```

Not pure Ada, but fits very well.

Thank to all suggestions.

From: Pablo Rego <pvrego@gmail.com>
Date: Wed, 9 Nov 2011 17:16:51 -0800
Subject: Re: Read Windows login username in Ada 95
Newsgroups: comp.lang.ada

> For a pure Ada version why not use Ada.Environment_Variables?

The problem is that Ada.Environment_Variables is an Ada 2005 package, cannot use it.

From: Adam Beneschan <adam@irvine.com>
Date: Wed, 9 Nov 2011 16:02:51 -0800
Subject: Re: Read Windows login username in Ada 95
Newsgroups: comp.lang.ada

> In Windows, you can call function GetEnvironmentVariable and get the value of the environment variable "USERNAME".

Doesn't always work (and neither does using Ada.Environment_Variables).

I just tried it and found that GetUserName returns my login name, while Ada.Environment_Variables says that "USERNAME" doesn't exist.

It may be an unusual setup--I'm logging into an Windows XP system remotely through the GoodTech telnet server. But you may as well use the function that works more reliably.

This worked for me, but it could use more error checking:

```
with Text_IO;
procedure Print_User_Name is
   subtype Buffer_Type is String
                          (1 .. 200);
   function GetUserName (
        lpBuffer : access Buffer_Type;
        lpnSize  : access Integer)
     return Integer;
   pragma Import (StdCall,
     GetUserName, "GetUserNameA");

   Buf : aliased Buffer_Type;
   Size : aliased Integer;
   Result : Integer;
begin
   Size := Buffer_Type'Length;
   Result := GetUserName (Buf'Access,
                          Size'Access);
   Text_IO.Put_Line (Buf (1 .. Size - 1));
end Print_User_Name;
```

StdCall is how we import Windows API functions with ICC Ada. Don't know how GNAT does it--probably the same. Result should be checked for errors, but I didn't bother. GetUserName sets Size to the size of the result including the null terminator, which is why the next line uses Size - 1.

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

April 03-05 — 4th **NASA Formal Methods Symposium** (NFM'2012), Norfolk, Virginia, USA. Topics include: identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems; formal verification, including theorem proving, model checking, and static analysis; model-based development; techniques and algorithms for scaling formal methods, such as abstraction and symbolic methods, parallel and distributed techniques, ...; code generation from formally verified models; significant applications of formal methods to aerospace systems; etc.

April 10-13 — 7th **European Conference on Computer Systems** (EuroSys'2012), Bern, Switzerland. Topics include: all areas of operating systems and distributed systems, including systems aspects of dependable computing, distributed computing, parallel and concurrent computing, programming-language support and runtime systems, real-time and embedded systems, security, etc.

☺ April 11-13 — 15th IEEE **International Symposium on Object/component/service-oriented Real-time distributed Computing** (ISORC'2012), Shenzhen, China. Topics include: Programming and system engineering (languages, model-driven development of high integrity applications, specification, design, verification, validation, maintenance, ...); System software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...); Applications (embedded systems (automotive, avionics, consumer electronics, etc), real-time object-oriented simulations, ...); System evaluation (timeliness, worst-case execution time, dependability, end-to-end QoS, fault detection and recovery time, ...); etc.

April 11-13 — 19th Annual **IEEE International Conference and Workshops on the Engineering of Computer Based Systems** (ECBS'2012), Novi Sad, Serbia. Topics include: Dependability, Safety, and Security; Distributed Systems Design & Architecture; ECBS Infrastructure (Tools, Platforms); Embedded Real-Time Software Systems; Model-based System Development; Verification & Validation; Reengineering & Reuse; Evolution & Change; etc.

April 17-19 — 25th **Conference on Software Engineering Education and Training** (CSEET'2012), Nanjing, China. Topics include: Technology Transfer; Student projects and internships; Industry-academia collaboration models; Software engineering professionalism; Education & training for "real-world" Software Engineering practices; Evaluation of SE Curricula: Are We Still Relevant?; Training models in industry; Systems and Software Engineering; Teaching the Business of Software Engineering; etc.

April 23-26 — 24th Annual **Systems and Software Technology Conference** (SSTC'2012), Salt Lake City, UT, USA.

May 08-11 — 9th **European Dependable Computing Conference** (EDCC'2012), Sibiu, Romania. Topics include: Hardware and software architecture of dependable systems, Safety critical systems, Embedded and real-time systems, Impact of manufacturing technology on dependability, Testing and validation methods, etc.

☺ May 21-25 — 26th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2012), Shanghai, China. Topics include: all areas of parallel and distributed processing, such as Parallel and distributed algorithms; Applications of parallel and distributed computing; Parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, parallel programming paradigms, programming environments and tools, etc.

☺ May 25 **Workshop on Multithreaded Architectures and Applications** (MTAAP'2012). Topics include: programming frameworks for multithreading in the form of languages and libraries, compilers, analysis and debugging tools to increase the programming productivity.

May 25 13th **International Workshop on Parallel and Distributed Scientific and Engineering Computing** (PDSEC-12). Topics include: parallel and distributed computing techniques and codes; practical experiences using various parallel and distributed systems; loop and task parallelism; scheduling; compiler issues for scientific and engineering computing; scientific and engineering computing on parallel computers, multicores, GPUs, FPGAs, ...; etc.

☺ May 29-31 50th **International Conference on Objects, Models, Components, Patterns** (TOOLS Europe'2012), Prague, Czech Republic. Topics include: Object technology, programming techniques, languages, tools; Language implementation techniques, compilers, run-time systems; Distributed and concurrent object systems, multicore programming; Program verification and analysis techniques; Trusted, reliable and secure components; Component-based programming, modeling, tools; Model-driven development; Empirical studies on programming models and techniques; Domain specific languages and language design; Industrial-strength experience reports; Real-time object-oriented programming and design; etc.

May 31- Jun 1 **International Conference on Multicore Software Engineering, Performance, and Tools** (MSEPT'2012). Topics include: from small-scale systems to large-scale parallel systems; from writing new applications to reengineering legacy applications; frameworks and libraries for multicore software; parallel software architectures; modeling techniques for multicore software; programming models for multicore; testing and debugging of parallel applications; verification techniques for multicore software; software reengineering for parallelism; development environments and tools for multicore software; compiler techniques and auto-parallelization on multicore; multicore software issues in scientific computing; multicore software on mobile and embedded devices; experience reports; etc.

☺ June 02-09 34th **International Conference on Software Engineering** (ICSE'2012), Zurich, Switzerland. Theme: "Sustainable Software for a Sustainable World". Deadline for early registration: April 22, 2012.

June 01 5th **Workshop on Refactoring Tools** (WRT'2012). Topics include: refactoring engines, program analyses for refactoring tools, tools for suggesting refactorings, medium- and large-scale refactorings (e.g., package- or component-level), refactoring for concurrency and parallelism, etc.

June 02-03 9th **International Working Conference on Mining Software Repositories** (MSR'2012). Topics include: mining of repositories across multiple projects; characterization, classification, and prediction of software defects based on analysis of software repositories; techniques to model reliability and defect occurrences; search techniques to assist developers in finding suitable components and code fragments for reuse, and software search engines; analysis of change patterns and trends to assist in future development; empirical studies on extracting data from repositories of large long-lived and/or industrial projects; mining execution traces and logs; etc.

☺ June 09 5th **Workshop on Exception Handling** (WEH'2012). Topics include: Exceptions in the software life-cycle (specifications, architectural design, modelling and programming, verification, debugging, testing, refactoring, variability management, static analysis, etc); Exception handling for and with new software artefacts (aspects, components, etc); Exception handling in today's applications (distributed, web-based, cloud, etc); Empirical studies of exception handling; Design patterns and anti-patterns, architectural styles, and good programming practice; etc.

June 07-08 4th USENIX **Workshop on Hot Topics in Parallelism** (HotPar'2012), Berkeley, CA, USA.

June 07-10 21st **International Workshop on Algebraic Development Techniques** (WADT'2012), Salamanca, Spain. Topics include: other approaches to formal specification; specification languages, methods, and environments; model-driven development; integration of formal specification techniques; quality assurance, validation, and verification; etc.

♦ June 11-15 17th **International Conference on Reliable Software Technologies - Ada-Europe'2012**, Stockholm, Sweden. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN.

☺ June 11-16 26th **European Conference on Object-Oriented Programming** (ECOOP'2012), Beijing, China. Topics include: all areas of object technology and related software development technologies, such as Analysis and design methods; Concurrent, parallel, distributed, and real-time systems; Language design and implementation; Modularity, aspects, features, components, services; Software development environments and tools; Static and dynamic software analysis; Type systems, formal methods; Software evolution; etc.

☺ June 13 **International Workshop on Languages for the Multi-core Era** (LaME'2012). Topics include: programming language support for concurrency; the development of innovative or improved concurrency models, languages, run-time systems, libraries and tools for multicore programming. Deadline for submissions: April 15, 2012 (regular papers), May 20, 2012 (position papers, programming challenge).

June 11-16 ACM SIGPLAN **Conference on Programming Language Design and Implementation** (PLDI'2012), Beijing, China. Events also includes: 3rd Workshop on Experimental Evaluation of Software and Systems in Computer Science (Evaluate), ACM SIGPLAN 7th Workshop on Programming Languages and Analysis for Security (PLAS), 2nd ACM SIGPLAN Software Security and Protection Workshop (SSP), etc.

☺ June 14 1st **Asia-Pacific Programming Languages and Compilers Workshop** (APPLC'2012). Topics include: Language designs and extensions; Static and dynamic analysis of programs; Domain-specific languages and tools; Type systems and program logics; Checking or improving the security or correctness of programs; Memory management; Parallelism, both implicit and explicit; Novel programming models; Debugging techniques and tools; Interaction of compilers and run-time systems with underlying systems; etc.

June 13-15 37th **USENIX Annual Technical Conference** (USENIX ATC'2012), Boston, MA, USA. Topics include: Distributed and parallel systems; Embedded systems; Reliability, availability, and scalability; Security, privacy, and trust; etc.

June 13-16 7th **International Federated Conferences on Distributed Computing Techniques** (DisCoTec'2012), Stockholm, Sweden. Includes the COORDINATION, DAIS, and FMOODS & FORTE conferences.

June 18-22 9th **International Conference on Integrated Formal Methods** (iFM'2012), Pisa, Italy. Topics include: the combination of (formal and semi-formal) methods for system development, covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.

June 21-22 **Symposium on Languages, Applications and Technologies** (SLATE'2012), Braga, Portugal. Topics include: Programming language concepts and methodologies; Design of novel language constructs and their implementation; Domain Specific Languages design and implementation; Programming tools; Programming, refactoring and debugging environments; Dynamic and static analysis: Program Slicing Compilation and interpretation techniques; Code generation and optimization; Runtime techniques and Memory management; etc.

June 25-27 11th **International Conference on Mathematics of Program Construction** (MPC'2012), Madrid, Spain. Topics of interest range from algorithmics to support for program construction in programming languages and systems, such as type systems, program analysis and transformation, programming-language semantics, security, etc.

June 25-28 **Federated Events on Component-Based Software Engineering and Software Architecture** (CompArch'2012), Bertinoro, Italy.

June 26-28 3rd **International Symposium on Architecting Critical Systems** (ISARCS'2012). Topics include: architectural support for evolution; automotive and avionic systems; component-based development; critical infrastructures; embedded, mobile, and ubiquitous systems; industrial case studies, challenges, problems, and solutions; integrators (wrappers) for dependability; model-driven development; runtime checks; survivability and error confinement; type checking techniques; etc.

June 27-29 — 12th **International Conference on Application of Concurrency to System Design** (ACSD'2012), Hamburg, Germany. Topics include: (industrial) case studies of general interest, gaming applications, automotive systems, (bio-)medical applications, internet and grid computing, etc.; synthesis and control of concurrent systems, (compositional) modeling and design, (modular) synthesis and analysis, distributed simulation and implementation, ...; etc.

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. Deadline for early registration: May 10, 2012.

☺ 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. Deadline for submissions: April 20, 2012 (fast abstracts, posters, doctoral symposium papers).

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. Deadline for early registration: May 30, 2012.

☺ 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". Deadline for submissions: April 23, 2012 (papers).

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. Deadline for application: April 13, 2012 (PhD Symposium).

☺ 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 approached 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.

☺ Sep 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.

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

Sep 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 method 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 submissions: May 25, 2012 (panels, tutorials). Deadline for early registration: June 15, 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: June 1, 2012 (full papers), 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. Deadline for submissions: May 20, 2012 (short papers, posters).

☺ Sep 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. Deadline for submissions: April 15, 2012 (research track abstracts), April 20, 2012 (research track), May 31, 2012 (doctoral symposium), June 25, 2012 (early research achievements track), June 27, 2012 (industry track, tool demo track).

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. Deadline for submissions: June 4, 2012 (abstracts), June 11, 2012 (papers).

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. Deadline for submissions: April 16, 2012 (abstracts), April 23, 2012 (papers).

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. Deadline for submissions: April 19, 2012 (papers).

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. Deadline for submissions: June 25, 2012 (workshop papers).

☺ 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: April 13, 2012 (OOPSLA research papers, Onward! papers, Onward! essays, Wavefront, Wavefront Experience, workshops, panels); July 9, 2012 (posters, ACM Student Research competition, Doctoral Symposium); July 11, 2012 (Dynamic Languages Symposium); July 15, 2012 (demonstrations).

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. Deadline for submissions: April 16, 2012 (full papers).

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 7, 2012.

♦ Dec 02-06 **ACM SIGAda Annual International Conference**, Boston, Massachusetts, USA.

December 05-07 33rd IEEE **Real-Time Systems Symposium** (RTSS'2012), San Juan, Porto Rico. RTSS provides a forum for the presentation of high-quality, original research covering all aspects of real-time systems design, analysis, implementation, evaluation, and experiences. Deadline for submissions: May 15, 2012.

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

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: May 16, 2012 (papers), September 16, 2012 (student symposium). Deadline for early registration: November 14, 2012.

2013

☺ January 20-22 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: April 22, 2012 (co-located events).

March 25-29 12th **International Conference on Aspect-Oriented Software Development** (AOSD'2013), Fukuoka, Japan. Topics include: Complex systems; Software design and engineering (evolution, economics, composition, methodology, ...); Programming languages (language design, compilation and interpretation, verification and static program analysis, formal languages, execution environments and dynamic weaving, ...); Varieties of modularity (model-driven development, generative programming, software product lines, contracts and components, ...); Tools (evolution and reverse engineering, crosscutting views, refactoring, ...); Applications (distributed and concurrent systems, middleware, runtime verification, ...); etc. Deadline for submissions: May 7, 2012 (round 1), July 23, 2012 (round 2), October 8, 2012 (round 3).

17th International Conference on Reliable Software Technologies

Ada-Europe 2012

11-15 June 2012, Stockholm, Sweden

www.ada-europe.org/conference2012

Advance Information

The 17th International Conference on Reliable Software Technologies (Ada-Europe 2012) will take place in Stockholm, Sweden. This conference is the latest in a series of annual international conferences started in the early 80's, under the auspices of, and organization by, Ada-Europe, the international organization that promotes the knowledge and use of Ada and reliable software in general into academia, research and industry.

Ada-Europe 2012 provides a unique opportunity for dialogue and collaboration between academics and industrial practitioners interesting in reliable software.

Following tradition, the conference will span a full week, including tutorials and a central three-day technical program with the latest advances in reliable software technologies and Ada. The core program features 3 keynote talks, 15 refereed scientific papers on topics in the conference theme, 9 industrial presentations and 2 discussion panels. Participants will have ample choice of half-day and full-day tutorials on Monday and Friday. Tutorials consist of courses given by recognised experts on topics concerning state-of-the-art methods and technologies for the development of reliable software. A session "Ada in Motion" is also planned to show off cases of Ada being used in moving equipment, such as Lego Mindstorms robots or Arduino based devices.

Program Highlights

Each day of the core program will open with a keynote talk delivered by one the following eminent speakers:

- Bertrand Meyer, ETH Zurich, Switzerland, Chief Architect of Eiffel Software, "*Life with Contracts*"
- Göran Backlund, Combitech, Sweden, "*What is the Mission of a Software Developer?*"
- Jean-Loup Terraillon, ESTEC/ESA, the Netherlands, "*Multicore Processors - the Next Generation Computer for ESA Space Missions*".

The program also features two discussion panels, scheduled in the afternoons of the Tuesday and the Wednesday of the conference week:

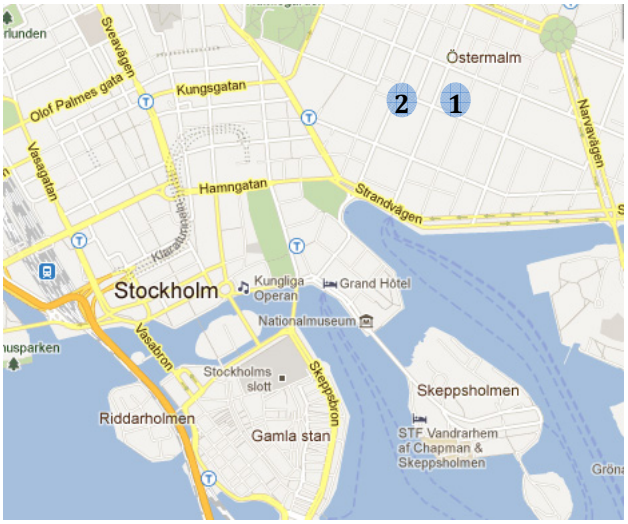
- Panel 1 (Tuesday): "*What is Language Technology in Our Time?*", with Tullio Vardanega (University of Padua) as moderator, in which the invited language specialists will discuss how in their view the role, nature and contents of language technology has currently become, what the drivers of the change have been and can be expected to be, and how Ada should respond to them.
- Panel 2 (Wednesday): "*Reliable Software, a Perspective from Industry*", with Jørgen Bundgaard (Rovsing A/S) as moderator, in which the invited panellists will discuss what they see as the most pressing and challenging industrial needs in the way of software technology to facilitate the production of reliable software.

Both panel sessions will allow and include open interaction with the conference participants.

About the Venue

Stockholm, one of the most beautiful capitals in the world, is built on 14 islands around one of Europe's largest and best-preserved mediaeval city centres, located by the Baltic Sea coast. Stockholm is also Scandinavia's financial center with the largest gross regional product and largest presence of international companies.

In 2010, Stockholm was the first city to receive the European Green Capital award, an initiative of the European Commission, and is ranked the fourth in the "Cities of Opportunity" analysis, ranking first in intellectual capital and innovation, health, safety and security demographics and liveability.



The Ada-Europe 2012 conference will take place at Näringslivets Hus ①, a modern conference centre situated in the very heart of Stockholm, located near the Östermalmstorg metro station and close to the Gamla Stan historic district.

The program of the conference will offer ample time for interaction and networking, with extensive lunch and coffee periods, and a banquet being held on Wednesday, at Östermalms Saluhall ②, a marketplace food hall in a magnificent building from 1888.

Ada-Europe 2012 will build on the success of the 2011 event, in Edinburgh, UK, on June 20-24, which attracted over 130 delegates coming from Belgium, Brazil, Canada, Denmark, Egypt, Finland, France, Germany, Israel, Italy, Norway, Poland, Portugal, Russia, Slovakia, South Africa, Spain, Sweden, Switzerland, The Netherlands, UK and USA, representing more than 20 universities and 50 companies.

Further Information

The conference website at `www.ada-europe.org/conference2012` provides full and up-to-date details of the program, venue and social program, registration, accommodation and travel advice.

For exhibiting and sponsoring details please contact the Conference Chair, Ahlan Marriott, at ahlan@ada-switzerland.ch.

For local information please contact the Local Chair, Rei Stråhle, at rei@ada-sweden.org.

SIGAda, SIGBED, SIGPLAN

ACM SIGAda Annual International Conference

High Integrity Language Technology
HILT 2012

Call for Technical Contributions

Developing and Certifying Critical Software

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)
- Teaching high integrity development
- Case studies of high integrity systems
- Real-time networking/quality of service guarantees
- 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

KINDS OF TECHNICAL CONTRIBUTIONS

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM *Ada Letters*. The Proceedings will be entered into the widely consulted ACM Digital Library accessible online to university campuses, ACM's 100,000 members, and the software community.

EXTENDED ABSTRACTS discuss current work for which early submission of a full paper may be premature. If your abstract is accepted, 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. Descriptions will be published in the final program or proceedings, but a paper will not be required.

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

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation and, if selected, a subsequent abstract for a 30-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for ACM *Ada Letters*.

WORKSHOPS are focused sessions that allow knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. 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**	Jeff Boleng, Program Chair jeff@boleng.com
Industrial presentation proposals	**August 1, 2012** (overview) **October 1, 2012** (abstract)	
Tutorial proposals	**June 29, 2012**	John McCormick, Tutorials Chair mccormick@cs.uni.edu

At least one author is required to register and make a presentation 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 Conference welcomes a grant application from anyone whose goals meet this description. The benefits include full conference registration with proceedings and registration costs for 2 days of conference tutorials/workshops. Partial travel funding is also available from AdaCore to faculty and students from GNAT Academic Program member institutions, which can be combined with conference grants. For more details visit the conference web site or contact Prof. Michael B. Feldman (MFeldman@gwu.edu)

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

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

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

ANY QUESTIONS?

Please send email to SIGAda.HILT2012@acm.org, or contact the Conference Chair (Ben Brosgol, brosgol@adacore.com), SIGAda's Vice-Chair for Meetings and Conferences (Alok Srivastava, asrivastava@yahoo.com), or SIGAda's Chair (Ricky E. Sward, rsward@mitre.org).

Rationale for Ada 2012: 2 Expressions

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 the introduction of more flexible forms of expressions in Ada 2012.

There are four new forms of expressions. If expressions and case expressions define a value and closely resemble if statements and case statements. Quantified expressions take two forms using for all and for some to return a Boolean value. Finally, expression functions provide a simple means of parameterizing an expression without the formality of providing a function body.

These more flexible forms of expressions will be found invaluable in formulating contracts such as preconditions. It is interesting to note that Ada now has conditional expressions over 50 years after their introduction in Algol 60.

Keywords: rationale, Ada 2012.

1 Overview of changes

One of the key areas identified by the WG9 guidance document [1] as needing attention was improving the ability to write and enforce contracts. These were discussed in detail in the previous paper.

When defining the new aspects for preconditions, postconditions, type invariants and subtype predicates it became clear that without more flexible forms of expressions, many functions would need to be introduced because in all cases the aspect was given by an expression.

However, declaring a function and thus giving the detail of the condition, invariant or predicate in the function body makes the detail of the contract rather remote for the human reader. Information hiding is usually a good thing but in this case, it just introduces obscurity.

Four forms are introduced, namely, if expressions, case expressions, quantified expressions and expression functions. Together they give Ada some of the flexible feel of a functional language.

In addition, membership tests are generalized to allow greater flexibility which is particularly useful for subtype predicates.

The following Ada issues cover the key changes and are described in detail in this paper:

3 Qualified expressions and names

147 Conditional expressions

158 Generalizing membership tests

176 Quantified expressions

177 Expression functions

188 Case expressions

These changes can be grouped as follows.

First there are conditional expressions which come in two forms, if expressions and case expressions, which have a number of features in common (147, 188).

Then there is the introduction of quantified expressions which use **for all** to describe a universal quantifier and **for some** to describe an existential quantifier. Note that **some** is a new reserved word (176).

Next comes the fourth new form of expression which is the expression function (177).

Finally, membership tests are generalized (158) and there is a minor change regarding qualified expressions (3).

2 If expressions

It is perhaps very surprising that Ada does not have if expressions as well as if statements. In order to provide some background context we briefly look at two historic languages that are perhaps the main precursors to modern languages; these are Algol 60 [2] and CPL [3].

Algol 60 had conditional expressions of the form

```
Z := if X = Y then P else Q
```

which can be contrasted with the conditional statement

```
if X = Y then
   Z := P
else
   Z := Q
```

Conditional statements in Algol 60 allowed only a single statement in each branch, so if several were required then they had to be grouped into a compound statement thus

```
if X = Y then
   begin
      Z := P;  A := B
   end
else
   begin
      Z := Q;  A := C
   end
```

It may be recalled that statements were not terminated by semicolons in Algol 60 but separated by them. However, a

null statement was simply nothing so the effect of adding an extra semicolon in some cases was harmless. However, accidentally writing

```
if X = Y then ;
  begin
    Z := P;  A := B
  end;
```

results in a disaster because the test then just covers a null statement and the assignments to Z and A always take place. The complexity of compound statements did not arise with conditional expressions.

The designers of Algol 68 [4] sensibly recognized the problem and introduced closing brackets thus

```
if X = Y then
  Z := P; A := B;
fi;
```

where **fi** matches the **if**. Conditional expressions in Algol 68 were similar

```
Z := if X = Y then P else Q fi;
```

An alternative shorthand notation was

```
Z := (X = Y | P | Q);
```

which was perhaps a bit too short.

The next major language in this series was Pascal [5]. The designers of Pascal rejected everything that had been learnt from Algol 68 and foolishly continued the Algol 60 style for compound statements and also dropped conditional expressions. Only with Modula did they realise the need for bracketing rather than compounding.

The other foundation language was CPL [3]. Conditional statements in CPL took the following form

```
if X = Y then do Z := P
```

```
if X = Y then do § Z := P; A := B §
```

where compound statements were delimited by section symbols (note that the closing symbol has a vertical line through it).

From CPL came BCPL, B and C. Along the way, the expressive := for assignment got lost in favour of = which then meant that = had to be replaced by == for equality. And the section brackets became { and } so in C the above conditional statements become

```
if (X == Y) Z = P;
```

```
if (X == Y) {Z = P;  A = B;}
```

This suffers from the same stray semicolon problem mentioned above with reference to Algol 60.

Steelman [6] did not require Ada to have conditional expressions and since they were not required they were not provided (the requirements were treated with considerable reverence). A further influence might have been that the new language had to be based on one of Pascal, Fortran and PL/I and Ada is based on Pascal which did not have conditional expressions as mentioned above.

Moreover, the Ada designers felt that the Algol 68 style with reversed keywords such as **fi** (or worse **esac**) for conditional statements would not be acceptable to the USDoD or the public at large and so we have **end if** as the closing bracket thus

```
if X = Y then
  Z := P;
  A := B;
end if;
```

Remember that semicolons terminate statements in Ada and so those above are all required. Moreover, since null statements in Ada have to be given explicitly, placing a stray semicolon after **then** gives a compiler error.

The absence of conditional expressions is a pain. It leads to unnecessary duplication such as having to write

```
if X > 0 then
  P(A, B, D, E);
else
  P(A, C, D, E);
end if;
```

where all parameters but one are the same. This can even lead to disgusting coding using the fact that Boolean'Pos(True) is 1 whereas Boolean'Pos(False) is 0. Thus (assuming that B and C are of type Integer) the above could be written as a single procedure call thus

```
P(A, Boolean'Pos(X>0)*B+Boolean'Pos(X<=0)*C, D, E);
```

So it is a great relief in Ada 2012 to be able to write

```
P(A, (if X>0 then B else C), D, E);
```

A worse problem was when a static expression was required such as the initial value for a named number as in the following gruesome code

```
Febdays: constant :=
     Boolean'Pos(Leap)*29 + Boolean'Pos(not Leap)*28;
```

which we can now thankfully write as

```
Febdays: constant := (if Leap then 29 else 28);
```

Note carefully that there is no **end if**. One reason is simply that it is logically unnecessary since there can be only a single expression after **else** and also **end if** would have been obtrusively heavy (compared say with **fi** of Algol 68). However, it was felt that some demarcation was required to aid clarity and so a conditional expression is always enclosed in parentheses. If the context already has parentheses then additional ones are not required. Thus in the case of a positional call with a single parameter we just write

```
P(if X > 0 then B else C);
```

but if we use named notation then extra parentheses are required

```
P(Para => (if X > 0 then B else C));
```

Note carefully that the term conditional expression in Ada 2012 embraces both if expressions and case expressions (which are discussed in the next section).

As expected, a series of tests can be done using **elsif** thus

```
P(if X > 0 then B elsif X < 0 then C else D);
```

and expressions can be nested

```
P(if X > 0 then (if Y > 0 then B else C) else D);
```

Without the rule requiring enclosing parentheses this could be written as

```
P(if X > 0 then if Y > 0 then B else C else D);  -- illegal
```

which seems more than a little confusing.

There is a special rule if the type of the expression is Boolean (that is of the predefined type Boolean or derived from it). In that case a final else part can be omitted and is taken to be true by default. Thus the following are equivalent

```
P(if C1 then C2 else True);
```

```
P(if C1 then C2);
```

Such abbreviations appear frequently in preconditions as was illustrated in the Introduction where we had

```
Pre => (if P1 > 0 then P2 > 0);
```

which has the obvious meaning that the precondition requires that if P1 is positive then P2 must be positive as well but if P1 is not positive then all is well and we don't care about P2.

This abbreviated form has the same effect as an implies operation.

```
R := C1 implies C2;      -- not Ada!
```

with the following truth table

	C1 = False	C1 = True
C2 = False	R = True	R = False
C2 = True	R = True	R = True

Some consideration was given to including such an operation in Ada 2012 (it existed in Algol 60). However, this is exactly the same as

```
R := not C1 or C2;
```

and so somewhat unnecessary. Moreover, although **implies** might appeal to some programmers it could lead to maintenance problems since it might be considered incomprehensible by many others.

There are important rules regarding the types of the various dependent expressions in the branches of an if expression. Basically they have to all be of the same type or convertible to the same expected type. But there are some interesting situations.

If the conditional expression is the argument of a type conversion then effectively the conversion is considered pushed down to the dependent expressions. Thus

```
X := Float(if P then A else B);
```

is equivalent to

```
X := (if P then Float(A) else Float(B));
```

As a consequence we can write

```
X := Float(if P then 27 else 0.3);
```

and it doesn't matter that 27 and 0.3 are not of the same type.

If the expected type is class wide, perhaps giving the initial value for a class wide variable V, then the individual dependent expressions have that same expected class wide type but they need not all be of the same specific type within the class. Thus we might write

```
V: Object'Class := (if B then A_Circle else A_Triangle);
```

where A_Circle and A_Triangle are objects of specific types Circle and Triangle which are themselves descended from the type Object.

If the expected type is a specific tagged type then various situations can arise regarding the various branches which are similar to the rules for calling a subprogram with several controlling operands. Either they all have to be dynamically tagged (that is class wide) or all have to be statically tagged. They might all be tag indeterminate in which case the conditional expression as a whole is also tag indeterminate.

Some obscure curiosities arise. Remember that the controlling condition for an if statement can be any Boolean type. Consider

```
type My_Boolean is new Boolean;

My_Cond: My_Boolean := ... ;

if (if K > 10 then X = Y else My_Cond) then   -- illegal
   ...
end if;
```

The problem here is that X = Y is of type Boolean but My_Cond is of type My_Boolean. Moreover, the expected type for the condition in the if statement is any Boolean type so it cannot make up its mind. We could overcome this foolishness by putting a type conversion around the if expression.

There are also rules regarding staticness. If all branches are static then a conditional expression as a whole is static as in the example of Febdays above. Thus the definition of a static expression has been extended to permit the inclusion of static conditional expressions.

The avid reader of the Reference Manual will find that the term *statically unevaluated* has been introduced. This relates to situations where expressions are not evaluated because a prior expression is static. Consider

```
X := (if B then P else Q);
```

If B, P and Q are all static then the conditional expression as a whole is static. If B is true then the answer is P and there is not any need to even look at Q. We say that Q is statically unevaluated and indeed it does not matter that if Q had been evaluated it would have raised an exception. Thus we might write

```
Average := (if Count = 0 then 0.0 else Total/Count);
```

and there is no risk of dividing by zero.

Similar rules regarding being statically unevaluated apply to short circuit conditions, case expressions, and membership tests.

As might be expected there are rules regarding access types and accessibility. The accessibility level of a conditional expression is simply that of the chosen dependent expression and thus (generally) determined dynamically.

Readers might feel that Ada has embarked on a slippery slope by introducing more flexibility thereby possibly damaging Ada's reputation for reliability. Certainly a number of additional rules have been required but from the users' point of view these are almost intuitive. It should be remembered that the difficulties in C stem from a combination of things

- that assignment is permitted as an expression,
- that integer values are used as Booleans,
- that null statements are invisible.

None of these applies to Ada so all is well.

3 Case expressions

Case expressions have much in common with if expressions and the two are collectively known as conditional expressions.

Thus given a variable D of the familiar type Day, we can assign the number of hours in a working day by

```
Hours := (case D is
              when Mon .. Thurs => 8,
              when Fri => 6,
              when Sat | Sun => 0);
```

A slightly more adventurous example involving nested if expressions is

```
Days := (case M is
      when September | April | June | November => 30,
      when February =>
            (if Year mod 100 = 0 then
              (if Year mod 400 =0 then 29 else 28)
                    else
              (if Year mod 4 = 0 then 29 else 28)),
      when others => 31);
```

The reader is invited to improve this!

Note the similarity to the rules for if expressions. There is no closing **end case**. Case expressions are always enclosed in parentheses but they can be omitted if the context already provides parentheses.

If M and Year are static then the case expression as a whole is also static. If M is static and equal to September, April, June or November then the value is statically known to be 30 so that the expression for February is statically unevaluated even if Year is not static. Note that the various choices are evaluated in order.

The rules regarding the types of the dependent expressions are exactly as for if expressions. Thus if the case expression is the argument of a type conversion then the conversion is effectively pushed down to the dependent expressions.

It is always worth emphasizing that an important advantage of case constructions is that they give a coverage check. Thus in the previous paper we had

```
subtype Pet is Animal
  with Static_Predicate =>
    (case Pet is
          when Cat | Dog | Horse => True,
          when Bear | Wolf => False);
```

which is much more reliable than

```
subtype Pet is Animal
  with Static_Predicate => Pet in Cat | Dog | Horse;
```

because when we add Rabbit to the type Animal, we are forced to include it in one branch of the case expression whereas it is all too easy to forget it using an if expression.

4 Quantified expressions

Another new form of expression in Ada 2012 is the quantified expression. The syntax is

```
quantified_expression ::=
   for quantifier loop_parameter_specification => predicate
   | for quantifier iterator_specification => predicate

quantifier ::= all | some

predicate ::= boolean_expression
```

The form involving iterator_specification concerns generalized iterators and will be found particularly useful with containers; it will be discussed in detail in a later paper. Here we will concentrate on the use of the familiar loop parameter specification.

The type of a quantified expression is Boolean. So we might write

```
B := (for all K in A'Range => A(K) = 0);
```

which assigns true to B if every component of the array A has value 0. We might also write

```
B := (for some K in A'Range => A(K) = 0);
```

which assigns true to B if some component of the array A has value 0.

Note that the loop parameter is almost inevitably used in the predicate. A quantified expression is very much like a for statement except that we evaluate the expression after => on each iteration rather than executing one or more

statements. The iteration is somewhat implicit and the words **loop** and **end loop** do not appear.

The expression is evaluated for each iteration in the appropriate order (**reverse** can be inserted of course) and the iteration stops as soon as the value of the expression is determined. Thus in the case of **for all**, as soon as one value is found to be False, the overall expression is False whereas in the case of **for some** as soon as one value is found to be True, the overall expression is True. An iteration could raise an exception which would then be propagated in the usual way.

Like conditional expressions, a quantified expression is always enclosed in parentheses which can be omitted if the context already provides them, such as in a procedure call with a single positional parameter.

Incidentally, predicate is a fancy word meaning Boolean expression. Older folk might recall that it also means the part of a sentence after the subject. Thus in the sentence "I love Ada", the subject is "I" and the predicate is "love Ada".

The forms **for all** and **for some** are technically known as the universal quantifier and existential quantifier respectively.

Note that **some** is a new reserved word (the only one in Ada 2012). There were five new ones in Ada 95 (**aliased**, **protected**, **requeue**, **tagged** and **until**) and three new ones in Ada 2005 (**interface**, **overriding** and **synchronized**). Hopefully we are converging.

The type of a quantified expression can be any Boolean type (that is the predefined type Boolean or perhaps My_Boolean derived from Boolean). The predicate must be of the same type as the expression as a whole. Thus if the predicate is of type My_Boolean then the quantified expression is also of type My_Boolean.

The syntax for quantified expressions uses => to introduce the predicate. This is similar to the established notation in SPARK [7]. Consideration was given to using a vertical bar which is common in mathematics but that would have been confusing because of its use in membership tests and other situations with multiple choices.

As illustrated in the Introduction, quantified expressions will find their major uses in pre- and postconditions. Thus a procedure Sort on an array A of type Atype such as

```
type Atype is array (Index) of Float;
```

might have specification

```
procedure Sort(A: in out Atype)
  with
    Post => A'Length < 2 or else
              (for all K in A'First .. Index'Pred(A'Last) =>
                    A(K) <= A(Index'Succ(K)));
```

where we are assuming that the index type need not be an integer type and so we have to use Succ and Pred. Note how the trivial cases of a null array or an array with a single component are dismissed first.

Quantified expressions can be nested. So we might check that all components of a two-dimensional array are zero by writing

```
B := (for all I in AA'Range(1) =>
        (for all J in AA'Range(2) => AA(I, J) = 0));
```

This can be done rather more neatly using the syntactic form

```
for quantifier iterator_specification => predicate
```

as will be discussed in detail in a later paper. We just write

```
B := (for all E of AA => E = 0);
```

which iterates over all elements of the array AA however many dimensions it has.

5 Expression functions

The final new form to be discussed is the expression function. As outlined in the Introduction, an expression function provides the effect of a small function without the formality of introducing a body. It is important to appreciate that strictly speaking an expression function is basically another form of function and not another form of expression. But it is convenient to discuss expression functions in this paper because like conditional expressions and quantified expressions they arose for use with aspect clauses such as pre- and postconditions.

The syntax is

```
expression_function_declaration ::=
   [overriding_indicator]
   function_specification is
     (expression)
     [aspect_specification] ;
```

As an example we can reconsider the type Point and the function Is_At_Origin thus

```
package P is
  type Point is tagged
    record
      X, Y: Float := 0.0;
    end record;

  function Is_At_Origin(P: Point) return Boolean is
    (P.X = 0.0 and P.Y = 0.0)
      with Inline;

  ...
end P;
```

The expression function Is_At_Origin is a primitive operation of Point just as if it were a normal function with a body. If a type My_Point is derived from Point then Is_At_Origin would be inherited or could be overridden with a normal function or another expression function. Thus an expression function can be prefixed by an overriding indicator as indicated by the syntax.

Expression functions can have an aspect clause and since by their very nature they will be short, this will frequently be **with** Inline as in this example.

The result of an expression function is given by an expression in parentheses. The parentheses are included to immediately distinguish the structure from a normal body which could start with an arbitrary local declaration. The expression can be any expression having the required type. It could for example be a quantified expression as in the following

```
function Is_Zero(A: Atype) return Boolean is
  (for all J in A'Range => A(J) = 0);
```

This is another example of a situation where the quantified expression does not need to be enclosed in its own parentheses because the context supplied by the expression function provides parentheses.

Expression functions can be completions as well as standing alone and this introduces a number of possibilities. Remember that many declarations require completing. For example an incomplete type such as

```
type Cell;                          -- an incomplete type
```

is typically completed by a full type declaration later on

```
type Cell is
  record ... end record;            -- its completion
```

Completion also applies to subprograms. Typically the declaration (that is the specification plus semicolon) of a subprogram appears in a package specification thus

```
package P is
  function F(X: T);                 -- declaration
  ...
end P;
```

and then the body of F which completes it appears in the body of P thus

```
package body P is
  function F(X: T) is               -- completion
  begin
    ...
  end F;
  ...
end P;
```

A function body cannot appear in a package specification. The only combinations are

function declaration F	function body F
in spec of P	in body of P
in body of P	in body of P
None	in body of P

Remember that mutual recursion may require that a body be given later so it is possible for a distinct declaration of F to appear in the body of P before the full body of F. In addition to the above the function body could be replaced by a stub and the proper body compiled separately but that is another story.

The rules regarding expression functions are rather different. An expression function can be declared alone as in the example of Is_At_Origin above; or it can be a completion of a function declaration and that completion can be in either the package specification or body. A frequently useful combination occurs with a private type where we need to make a function visible so that it can be used in a precondition and the expression function then occurs in the private part as a completion thus

```
package P is
  type Point is tagged private;
  function Is_At_Origin(P: Point) return Boolean
    with Inline;
  procedure Do_It(P: in Point; ... )
    with Pre => not Is_At_Origin;

private

  type Point is tagged
    record
      X, Y: Float := 0.0;
    end record;

function Is_At_Origin(P: Point) return Boolean is
  (P.X = 0.0 and P.Y = 0.0);

  ...
end P;
```

Note that we cannot give an aspect specification on an expression function used as a completion, so it is given on the function specification; this makes it visible to the user. (This rule applies to all completions such as subprogram bodies and is not special to expression functions.)

An expression function can also be used in a package body as an abbreviation for

```
function Is_At_Origin(P: Point) return Boolean is
begin
  return P.X = 0.0 and P.Y = 0.0;
end Is_At_Origin;
```

The possible combinations regarding a function in a package are

function declaration F	expression function F
in spec of P	in spec or body of P
in body of P	in body of P
None	in spec or body of P

We perhaps naturally think of an expression function used as a completion to be in the private part of a package. But we could declare a function in the visible part of a package and then an expression function to complete it in the visible part as well. This is illustrated by the following interesting example of two mutually recursive functions.

```
package Hof is

   function M(K: Natural) return Natural;
   function F(K: Natural) return Natural;

   function M(K: Natural) return Natural is
     (if K = 0 then 0 else K – F(M(K–1)));

   function F(K: Natural) return Natural is
     (if K =0 then 1 else K – M(F(K–1)));

end Hof;
```

These are the Male and Female functions described by Hofstadter [8]. They are inextricably intertwined and both are given with completions for symmetry.

Almost inevitably, at least one of the expression functions in a mutually recursive pair will include an if expression (or else **or else**) otherwise the recursion will not stop.

Expression functions can also be declared in subprograms and blocks (they are basic declarative items). Moreover, an expression function that completes a function can also be declared in the subprogram or block.

This is illustrated by the following Gauss-Legendre algorithm which computes π to an amazing accuracy determined by the value of the constant K.

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Long_Long_Float_Text_IO;
use Ada.Long_Long_Float_Text_IO;
with Ada.Numerics.Long_Long_Elementary_Functions;
use Ada.Numerics.Long_Long_Elementary_Functions;
procedure Compute_Pi is

   function B(N: Natural) return Long_Long_Float;

   function A(N: Natural) return Long_Long_Float is
     (if N = 0 then 1.0 else (A(N–1)+B(N–1))/2.0);

   function B(N: Natural) return Long_Long_Float is
    (if N = 0 then Sqrt(0.5) else Sqrt(A(N–1)*B(N–1)));

   function T(N: Natural) return Long_Long_Float is
     (if N = 0 then 0.25 else
        (T(N–1)–2.0**(N–1)*A(N–1)–A(N))**2);

   K: constant := 5;                    -- for example
   Pi: constant Long_Long_Float :=
                  ((A(K) + B(K))**2 / (4.0*T(K));
begin
   Put(Pi, Exp => 0);
   New_Line;
end Compute_Pi;
```

With luck this will output 3.14159265358979324 (depending on the accuracy of Long_Long_Float).

The functions A and B give successive arithmetic and geometric means. They call each other and so B is given as a function specification which is then completed by the function expression.

I am grateful to Brad Moore and to Ed Schonberg for these instructive examples.

The rules regarding null procedures (introduced in Ada 2005 primarily for use with interfaces) are modified in Ada 2012 to be uniform with those for expression functions regarding completion. Thus

```
procedure Nothing(X: in T) is null;
```

can be used alone as a declaration of a null operation for a type or as a shorthand for a traditional null procedure thus possibly completing the declaration

```
procedure Nothing(X: in T);
```

Expression functions and null procedures do not count as subprogram declarations and so cannot be declared at library level. Nor can they be used as proper bodies to complete stubs. Library subprograms are mainly intended for use as main subprograms and to use an expression function in that way would be somewhat undignified!

Thus if we wanted to declare a useful function to compute sin 2*x* from time to time, we cannot write

```
with Ada.Numerics.Elementary_Functions;
use Ada.Numerics.Elementary_Functions;
function Sin2(X: Float) is                          -- illegal
  (2.0 * Sin(X) * Cos(X));
```

but either have to write it out the long way or wrap the expression function in a package.

6 Membership tests

Membership tests in Ada 83 to Ada 2005 are somewhat restrictive. They take two forms

- to test whether a value is in a given range, or
- to test whether a value is in a given subtype.

Examples of these are

```
if M in June .. August then

if I in Index then
```

However, the restrictions are annoying. If we want to test whether it is safe to eat an oyster (there has to be an R in the month) then we would really like to write

```
if M in Jan .. April | Sep .. Dec then -- illegal in Ada 2005
```

whereas we are forced to write something like

```
if M in Jan .. April or M in Sep .. Dec then
```

which means repeating M and then perhaps worrying about whether to use **or** or **or else**. Or in this case we could do the test the other way

```
if M not in May .. Aug then
```

What we would really like to do is use the vertical bar as in case statements and aggregates to select a combination of ranges, subtypes, and values.

Ada 2012 is much more flexible in this area. To see the differences it is probably easiest to look at the old and new syntax. The relevant old syntax for Ada 2005 is

relation ::=
 simple_expression [relational_operator simple_expression]
| simple_expression [**not**] **in** range
| simple_expression [**not**] **in** subtype_mark

where the last two productions define membership tests. The syntax regarding choices in aggregates and case statements in Ada 2005 is

discrete_choice_list ::= discrete_choice { | discrete_choice}

discrete_choice ::= expression | discrete_range | **others**

discrete_range ::= *discrete*_subtype_indication | range

The syntax in Ada 2012 is rather different and changes relation to use new productions for membership_choice_list and membership_choice (this enables the vertical bar to be used in membership tests). And then membership_test in turn uses choice_expression and choice_relation as follows

relation ::=
 simple_expression [relational_operator simple_expression]
| simple_expression [**not**] **in** membership_choice_list

membership_choice_list ::=
        membership_choice { | membership_choice}

membership_choice ::=
        choice_expression | range | subtype_mark

choice_expression ::=
    choice_relation {**and** choice_relation}
  | choice_relation {**or** choice_relation}
  | choice_relation {**xor** choice_relation}
  | choice_relation {**and then** choice_relation}
  | choice_relation {**or else** choice_relation}

choice_relation ::=
 simple_expression [relational_operator simple_expression]

The difference between a choice_relation and a relation is that the choice_relation does not include membership tests. Moreover, discrete_choice is changed to

discrete_choice ::= choice_expression
                | *discrete*_subtype_indication | range | **others**

the difference being that a discrete_choice now uses a choice_expression rather than an expression as one of its possibilities.

The overall effect of the changes is to permit the vertical bar in membership tests without getting too confused by nesting membership tests.

Here are some examples that are now permitted in Ada 2012 but were not permitted in Ada 2005

```
if N in 6 | 28 | 496 then          -- N is small and perfect!

if M in Spring | June | October .. December then
        -- combination of subtype, single value and range

if X in 0.5 .. Z | 2.0*Z .. 10.0 then  -- not discrete or static

if Obj in Triangle | Circle then      -- with tagged types

if Letter in 'A' | 'E' | 'I' | 'O' | 'U' then          -- characters
```

Membership tests are permitted for any type and values do not have to be static. There is no change here but it should be remembered that existing uses of the vertical bar in case statements and aggregates do require the type to be discrete and the values to be static.

Another important point about membership tests is that the membership choices are evaluated in order and as soon as one is found to be true (or false if **not** is present) then the relation as a whole is determined and the other membership choices are not evaluated. This is therefore the same as using short circuit forms such as **or else** and so gives another example of expressions which are statically unevaluated.

There is one very minor incompatibility. In Ada 2005 we can write

```
X: Boolean := ...
case X is
   when Y in 1 .. 10  => F(10);
   when others => F(5);
end case;
```

This is rather peculiar. The discrete choice Y **in** 1 .. 10 must be static. Suppose Y is 5, so that Y **in** 1 .. 10 is true; then if X is True, we call F(10) whereas if X is false we call F(5). And vice versa for values of Y not in the range 1 to 10.

This is syntactically illegal in Ada 2012 because a discrete choice can no longer be an expression and so be a membership test. This was imposed because otherwise we might have been tempted to write

```
X: Boolean := ...
case X is
   when Y in 1 .. 10 | 20 => F(10);
   when others => F(5);
end case;
```

and this is syntactically ambiguous because it might be parsed as (Y **in** 1 .. 10) | 20 rather than Y **in** (1 .. 10) | 20. Although it would be rejected anyway because of the type mismatch. However, syntactic ambiguities are disliked in Ada.

This is clearly very unlikely to be a problem. Case statements over Boolean types are pretty rare anyway.

7 Qualified expressions

We conclude this discussion of expressions by considering some points regarding names and primaries.

In Ada 2005 we have

name ::=
    direct_name | explicit_dereference | indexed_component
  | slice | selected_component | attribute_reference
  | type_conversion | function_call | character_literal

primary ::=
    numeric_literal | **null** | string_litreral | aggregate | name
  | qualified_expression | allocator | (expression)

And in Ada 2012 we have

```
name ::=
    direct_name | explicit_dereference | indexed_component
  | slice | selected_component | attribute_reference
  | type_conversion | function_call | character_literal
  | qualified_expression | generalized_reference
  | generalized_indexing

primary ::=
    numeric_literal | null | string_litreral | aggregate | name
  | allocator | (expression)
  | (conditional_expression) | (quantified_expression)
```

The important thing to observe here is that qualified_expression has moved from being a form of primary to being a name.

We also note the addition of conditional_expression and quantified_expression (both in parentheses) as forms of primary as discussed earlier in this paper and the addition of generalized_reference and generalized_indexing as forms of name. These are used in the new forms of iterator briefly alluded to at the end of the discussion on quantified expressions and which will be discussed in a later paper.

Returning to qualified expressions, the main reason for allowing them as names is to avoid unnecessary conversions as mentioned in the Introduction.

Consider

```
A: T;                                -- object of type T
type Art is array (1 .. 10) of T;    -- array of type T
function F(X: Integer) return Art;
```

A function call can be used as a prefix and so a call returning an array can be indexed as in

```
A := F(3)(7);
```

which assigns to A the value of the 7th component of the array returned by the call of F.

Now suppose that F is overloaded so that F(3) is ambiguous. The normal solution to such ambiguities is to use qualification and write Art'(F(3)) as in

```
A := Art'(F(3))(7);          -- illegal in Ada 2005
```

but this is illegal in Ada 2005 because a qualified expression is not a name and so cannot be used as a prefix. What one has to do in Ada 2005 is either copy the wretched array (really naughty) or add a type conversion (a type conversion *is* a name) thus

```
A := Art(Art'(F(3)))(7);
```

This is really gruesome; but in Ada 2012, qualification is permitted as a name so we can simply write

```
A := Art'(F(3))(7);          -- OK in Ada 2012
```

Although a qualified expression is now classed as a name rather than a primary, a qualified variable is not considered to be a variable. As a consequence, a qualified variable cannot be used as the destination of an assignment or as an actual parameter corresponding to an **out** or **in out** parameter. This would have added complexity for no useful purpose. Ambiguity generally involves calls on overloaded functions, and the result of a function call is always a constant, so ambiguous names of variables are unlikely!

Other uses might involve strings which can also give rise to ambiguities. For example

```
("a string")'Length
```

is ambiguous (it could be a String or Wide_String). But now we can write

```
String'("a string")'Length
```

which was not permitted in Ada 2005.

References

[1] ISO/IEC JTC1/SC22/WG9 N498 (2009) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of Amendment 2 to ISO/IEC 8652*.

[2] P. Naur (ed.), *Revised Report on the Algorithmic Language ALGOL 60* (1963) Communications of the Association for Computing Machinery, Vol. 6, p. 1.

[3] D. W. Barron et al (1963) *The main features of CPL*, Computer Journal vol. 6, pp 134-143.

[4] A. van Wijngaarden et al (eds) (1973) *Revised Report on the Algorithmic Language – ALGOL 68*, Springer-Verlag.

[5] K. Jensen and N. Wirth (1975) *Pascal User Manual and Report*, Springer-Verlag.

[6] Defense Advanced Research Projects Agency (1978) *Department of Defense Requirements for High Order Computer Programming Languages – STEELMAN*, USDoD.

[7] J. G. P. Barnes (2003) *High Integrity Software, The SPARK Approach to Safety and Security*, Addison-Wesley.

[8] D. R. Hofstadter (1980) *Gödel, Escher, Bach: an Eternal Golden Braid*, Basic Books.

Entity-Life Modeling: Designing Reactive Software Architectures to the Strengths of Tasks

Bo I Sandén
Colorado Technical U., 4435 N. Chestnut St., Colorado Springs, CO 80907, USA; email: bsanden@acm.org

Abstract

Entity-life modeling (ELM) is a design approach for multitask reactive software, which must respond to events in the environment as they occur. The abundant computing power now afforded by multiprocessors allows us to design such software differently than in the past. With ELM, the architect identifies threads of event occurrences in the problem domain that unfold independently or nearly so, and bases tasks on such event threads. ELM also provides design patterns for modeling tasks on activities defined for state machines, and event-thread patterns for problems that involve the sharing of problem-domain resources.

Keywords: multitasking, entity-life modeling, multiprocessors, event threads, resource sharing, design of multitask software, multithreading, task architecture, reactive systems.

1 Introduction

Tasking has long been used in certain kinds of software such as servers and real-time systems. The new abundance of processors allows us to design such software in more elegant ways that makes it more understandable and maintainable.

We are interested here in systems that exhibit a reactive behavior [12]. Their reactions to events in the problem domain can be captured in state models. Taken in a broad sense, reactive systems include telephone switches, embedded control systems, and interactive systems ranging from ATMs to travel-reservation systems [25].

This article presents entity-life modeling (ELM) as a design approach for reactive software [19, 20, 22]. ELM proposes certain architectural styles and suggests heuristics for finding task architectures in those styles. As there is often little time for upfront designing, ELM produces a task architecture directly, without first modeling objects or mapping out a data flow. ELM does not address the parallelization of algorithms but in no way precludes it; tasks computing concurrently on different processors are very much in ELM's spirit.

While the book *Design of Multithreaded Software* [22] defines ELM, this article applies the design approach to tasking in Ada and uses Ada terminology. It also introduces a UML diagram showing ELM's conceptual apparatus (Figure 1), which is not found in the book. The term "essential event" is introduced. The garage-door example in section 2 is new as well.

1.1 Traditional task architectures

Traditionally, multitask reactive software has been designed around the need to husband limited processor resources. It is often structured as a set of periodic tasks with periodic, *hard deadlines*, and each task's priority is set to ensure that it meets its deadlines. Schemes such as rate-monotonic scheduling can establish a priori whether all deadlines will be met [13]. The number of missed deadlines is used as a performance metric. Each input/stimulus visits one periodic task after the other, and may itself have a hard deadline. The periodic deadlines must be chosen so that deadline is also met.

Besides hard deadlines, we also talk about *soft deadlines*. Missing a soft deadline is no disaster but may degrade performance. In a cruise controller for example, which must adjust the throttle at consistent intervals for a smooth ride, the planned time for each adjustment can be a soft deadline [21, 22]. And interactive systems must be responsive to human input but again without hard deadlines. In such systems the mean and variance of the response or service time may be the best performance metric.

With additional cores and processors, inputs can still have associated deadlines, but there is much more processing time to go around: As many computations as there are processors or cores can proceed simultaneously. For many applications, the design need no longer center on processor availability as the overshadowing constraint.

This does not mean that we can blithely assume that abundant processes and cores will make all resource conflicts just go away. Even if an input is processed on a dedicated processor, it can miss its deadline because other tasks hold on to a shared resource too long. And in some "cyber-physical systems", computation may be "deeply embedded in and interacting with physical processes" imposing precise timing constraints on the software [16].

Thus, even absent the need for each input to visit one task after another, we must design multitask reactive software carefully. Undisciplined tasking can create "wild nondeterminism" and deadlock [15]. While a sequential program can be checked against an exhaustive set of test cases this is not so with tasking where some bugs may show up only in rare runtime situations that a tester cannot easily recreate. No less important, there are also general

software-engineering goals such as simplicity, modifiability, and maintainability.

1.2 Entity-life modeling

Entity-life modeling (ELM) is a disciplined design approach for multitask reactive software. It is dedicated to the proposition that each essential event occurrence should be processed to completion by a single task – or by an event/interrupt handler. This does not mean that a new task is started for each event occurrence. Having processed one to completion, a task can go on to handle another occurrence of the same or a different event and indeed a whole series of occurrences provided they are not too close together. For this, ELM defines an **event thread** as a chronological sequence of problem-domain event occurrences that are separated in time.[1]

The term "event" is taken from state modeling. In a garage-door controller, *click* may be the event where a homeowner hits a clicker button, and *top* may be where the door hits the ceiling. Each event has any number of occurrences, as when the owner of a particular home clicks the button at a particular time. Such an occurrence is normally followed by an occurrence of *top* seconds later.

ELM stipulates that every task must be based on an event thread. This justifies the task's existence. But if all the thread's occurrences are processed by handlers, no task is needed.

An ELM architecture is meant to be independent of the number of processors available. Thus it can use as many processors as there are tasks, or, at the other extreme, all tasks can run on a single processor. Clearly, the performance can be quite different but the architecture does not have to change.

The relationship between event-thread models and task architectures is illustrated schematically in Figure 1, which also shows that tasks have priorities. ELM assumes that their scheduling is *preemptive* so that a higher-priority task in need of a processor can take it over from a lower-priority task immediately.

1.2.1 Event-thread models and task architectures

Figure 1 shows an **event-thread model** of a given problem as a set of event threads. The threads partition the set of problem-domain event occurrences: Each occurrence belongs to exactly one event thread. There may be multiple ways to view a problem, each with its own event-thread model.

Also shown in Figure 1, a *task architecture* consists of *tasks* and *protected objects*. It is often called the *process* or *concurrency view* of the full software architecture [7, 14]. ELM uses "task architecture" both for brevity and to emphasize that it can be quite an independent artifact. It is self-contained because – except for event/interrupt handling – every instruction must be executed by a task. Each ELM task architecture is based on an event-thread model.

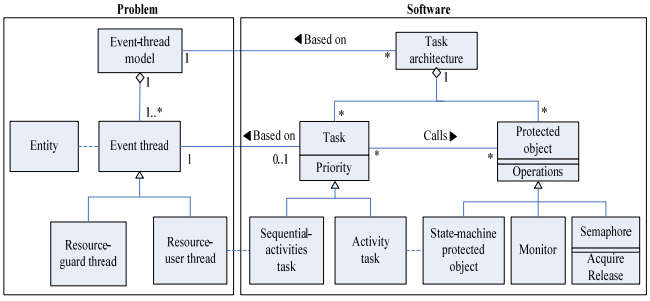


Figure 1 Basic ELM concepts shown as a UML class diagram

A protected object has *protected operations* with built-in *exclusion synchronization:* each task *locks* the object before operating on it. Because an object that is locked for any length of time can become a bottleneck, every protected operation in a reactive system should be nearly instantaneous. Thus no task should keep a protected object locked while performing a long computation. If all operations cannot be short, the design must account for wait states explicitly (2.1). Handlers for interrupts and timing events are protected procedures in Ada.

Protected objects can also have *entries*, which provide *condition synchronization*: A protected entry let tasks block on a condition [1]. ELM uses protected objects in various ways. Thus *monitor* and *semaphore* protected objects use condition synchronization to control the access to problem-domain resource (3.1, 4.3). We return to *state-machine* protected objects in 2.3. Protected objects can also be used as pragmatic engineering devices.

1.2.2 Event threads and entities

It is not enough for a set of event threads to partition the occurrences in a problem domain; we must also ensure that the event-thread model and the task architecture make sense. For that we need to find event threads that are intuitively meaningful and easy to grasp and describe. Here are some examples:

- In a garage-door controller for a home [2], the process of opening and closing defines an event thread where the user clicks a remote controller, the door reaches the top or bottom of its frame, etc. The control software is built around a single main state machine.
- The cruise controller [21, 22] also has a single main state machine. It has one event thread related to the human *driver*. In addition, it has a thread of time events governing the actions on the throttle. (A **time event** is the event that a certain time has passed since some other event occurrence.)
- In an elevator bank in a hotel or office building, each cabin is an instance of an *elevator* entity type. It travels up and down, stops, opens and closes its doors, etc.
- In a flexible manufacturing system (FMS), *jobs* visit workstations according to their process plans. Much of a job's life consists of waiting for exclusive access to a workstation or a vehicle (Section 4).

[1] How far apart the occurrences must be is a matter of engineering judgment [22].

As a matter of heuristics, it is often useful to look for **entities** in the problem domain such that each entity's *life history* is an event thread. In the examples above, a *car driver*, an *elevator*, and a *job* are such entities (or entity types). Naming an event thread after an entity creates a common set of expectations of how it should work so if entities are carefully chosen, we can agree on what belongs in each event thread. While an entity such as *car driver* can be helpful intuitively, it must also be very clear that we are only talking about the driver's interaction with a particular software system and no other doings. Because entities are auxiliary, the association with *event thread* is shown informally in Figure 1.

2 Analysis and design

In ELM analysis and design, event-thread modeling logically comes first and leads to a task architecture. The process is never linear, but an idealized flow is as follows:

1. Consider one or more possible *event-thread models*, each including all event occurrences in the problem domain. Capture event threads in state diagrams as needed (2.1).
2. *Validate* the feasibility of each model by ensuring that it covers all event occurrences in the problem (4.2, 5.1). Choose one model.
3. Realize the event-thread model by implementing each thread either as a task or by means of event/interrupt handling (2.3).

The next subsections elaborate on these steps.

2.1 Identifying event-thread models

We start by discussing how to identify event threads in a single state diagram. Entities stand out more clearly in complex problems, but there too, the life of an entity or entity type can be captured in a state diagram.

Figure 2 is a state diagram of an automated garage door for a home [2]. It shows how the complete system with software embedded must react to various events created by a user clicking a remote and by sensors around the physical door.

At each point in time, the door exists in exactly one *state* such as *Closed* (marked as the initial state) or *Opening*. In *Stopped-opening* and *Stopped-closing,* the door has been halted partway up or down. *Sensing* is a *super*state with the *sub*states *Closing*, *Stopped-opening* and *Opened*.

An *event* such as *click* can cause a *transition* to another state. The event *break* is when a light beam near the floor is broken; *bottom* is when the door reaches its closed position, etc. (We assume for now that *click*, *break*, *top*, and *bottom* create interrupts.)

Besides state transitions, events can also trigger *actions*. Thus *click/stop* indicates that–in certain states–the event *click* stops the door motor. The time event *S sec* in state *Stopped-closing* triggers automatic opening after the door has been still for *S* seconds.

Conceptually, an action is instantaneous, that is, it takes no time. For purposes of software design, it is *nearly* instantaneous, that is, short enough that no events happen *during* the action. What is short enough is an engineering-judgment call.

Many an action impacts the problem domain irreversibly and cannot be undone readily; for instance, *start down* could make the door crash into some obstacle.

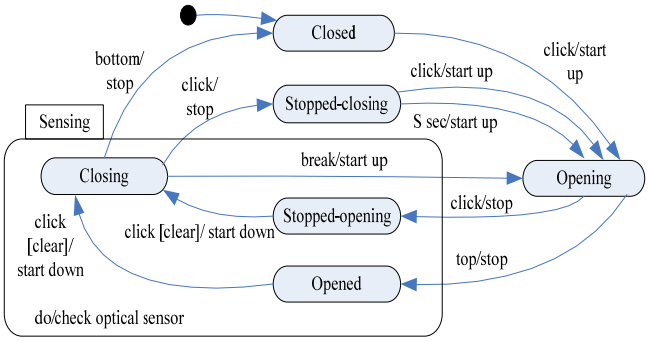


Figure 2 State diagram of a home garage door

State transitions and actions can be conditional: In Figure 2, *click [clear]/start down* triggers the action *start down* and a transition to *Closing* only if the light beam across the garage-door frame indicates that the passage is clear. (For simplicity, the diagram omits events and event-condition combinations causing neither transitions nor actions.)

If the system's reaction to an event is not nearly instantaneous it must be considered an *activity*, not an action. An activity continues throughout a state. It is indicated by the keyword *do* as with *check optical sensor* in the superstate *Sensing*: Throughout the superstate, the sensor is checked say every P seconds to ensure that the passage is clear. That activity starts immediately as the superstate is entered and stops upon the occurrence of an event – such as *break* – that causes a transition from *Sensing*.

In general, other possible activity types include lengthy computations. Also, ELM considers the *wait for a resource* an activity. This is because an activity is often implemented by a task, which can block on a protected object until the resource becomes available.

It is necessary in ELM to distinguish between "software" activities and others. A *software activity* is one that requires software involvement throughout its state; a lengthy computation and a wait for a resource are examples. Software activities require tasks.

Other activities are called *nominal* and consist of discrete actions that can be dealt with by interrupt/event handlers. An activity such as *check optical sensor* is nominal if it is handled by means of timing events, and a software activity if implemented as a task.

2.1.1 Essential events

Because ELM uses event threads to justify the task architecture, the events in the threads must be *essential* to the problem and not figments of a particular design. Most

essential events occur in the problem domain and are *shared* with the software; *click*, *break*, *top*, and *bottom* are examples. Time events such as *S sec* are essential if they are significant in the problem domain no matter the software design. Time events that trigger the sampling of problem-domain quantities are essential, for example.

Some essential events occur in the software. If the software controls domain-resource sharing they include *allocation events*, which are where a domain entity acquires a resource. The completion of a lengthy computation that is necessary no matter the design is also an essential event.

Note. "Essential event" is a more general term than "shared event" but is not used in the book [22]. It is clear, however, that some essential events are *not* intuitively shared. An allocation event, for example, is not shared with the domain, especially if the resource user simply goes on to wait for another resource.

2.2 Identifying event threads and thread models

In straightforward cases such as the garage door, a single state model covers the whole problem, and the event threads can be identified from the state diagram. For example, we might let a thread *clicks* include all occurrences of *click*, another, *bottoms*, include the occurrences of *bottom*, and a third, *tops*, the occurrences of *top*. The various *click* occurrences are indeed separated in time as are the occurrences of *bottom* and *top*. A fourth thread, *timer*, can include the occurrence of the event *S sec*.

It is immediately clear though that these event threads are interdependent. In ELM terms, the threads *bottoms* and *tops* do not *co-occur*. Here is a definition: *Two or more event threads in a thread model* ***co-occur*** *iff we can find an arbitrarily short time interval where each of them may have an event occurrence.* For practical purposes, event threads co-occur if there is a time when, by chance, each can have an event occurring [20, 22]. ELM recommends that the designer optimize the event-thread model by combining non-co-occurring threads. Thus in the garage door, one event thread *door-operation* can include the occurrences of *top*, *bottom*, and *S sec*.

The event thread *clicks* presents a little practical problem. While it does not co-occur with *door-operation* normally, a user could certainly hit the button just as the door reaches the top or bottom. Because this situation is exceptional we include the event *click* in *door-operation*. We could also keep *clicks* as a separate event thread, but ELM is not out to force more threads on the designer or make too much of some trivial aspect of the problem.

Another event thread, *sensor-checking*, is associated with the activity *check optical sensor* where the sensor is polled every P seconds. The time event *P sec* is essential, and its occurrences form the event thread. The threads *door-operation* and *sensor-checking* co-occur: the sensor might be polled at the same time as the user clicks the remote or the door hits the floor or ceiling.

An **optimal** event-thread model is one where all the threads co-occur. Its threads are as many as the maximum number of events that can ever occur at once. This number is the **concurrency level** of the problem. The garage-door problem's concurrency level is two, so a model consisting of the threads *door-operation* and *sensor-checking* is optimal. (Strictly, this is true only in the superstate *Sensing*; elsewhere the level is one.)

ELM does not *require* optimality because the concurrency level cannot always be determined readily. Besides, the designer is free to choose a nonoptimal model that is useful and intuitive even if some tasks should spend most of their time waiting for each other. Still, the concurrency level often has value as a benchmark. For example, if someone should propose a garage-door solution with five tasks, it is reasonable to ask whether five event occurrences need ever be handled at once.

2.3 Realizing an event-thread model in software

In an architecture based on an event-thread model, event threads can be implemented in two different ways: according to the **concurrent-activities** pattern or according to the **sequential-activities** pattern. (These are called *design* patterns [22], which seems consistent with the pattern community's usage [8], but "*implementation* patterns" might be intuitively clearer.)

In **the concurrent-activities pattern,** a **state-machine protected object** (Figure 1) keeps the current state, typically in a private variable. When an event occurs, an *event handler* is called. It is one of the object's operations, which updates the state variable as necessary and takes any action triggered by the event.

An **activity task** is associated with a state-machine protected object. It implements a software activity (or two or more non-co-occurring activities). In addition to event handlers, it can call other operations on the state-machine object to query the current state or to block until a certain state is entered. Each co-occurring activity needs its own activity task. In Figure 1, a dashed line informally shows that activity tasks are associated with state-machine objects. If there are no activities at all, the protected object alone represents the state machine. No tasks are needed.

The garage-door problem fits this pattern. Its state-machine protected object has handlers for *click*, *top*, *bottom*, and *break*, and a timing-event handler for *S sec*. The polling of the optical sensor has two possible implementations:

- A single, periodic activity task, *check-sensor*, blocks on the state-machine object until the superstate *Sensing* is entered. Within that state, it calls *break* to report each light-beam breach.
- The state-machine protected object has a timing-event handler for *P sec*. This architecture degenerates to a protected object without tasks.

The sequential-activities pattern works for activities that do *not* co-occur. They are arranged one after the other in a single **sequential-activities task** (Figure 1). It differs from an activity task in that it keeps track of the state, which can often be *implicit* in the logic rather than kept in a state

variable. Such task logic can read neatly from top to bottom like a main program, and can rely on the programming language's block structure, scoping rules, and exception handling as appropriate. Variables are allocated on the task's stack.

Like other patterns, those of ELM can "dovetail and intertwine" [8] in many ways. For instance, an activity task for a *superstate* can also be a sequential-activities task and keep track of the current substate within the superstate. This happens in the cruise controller. It has a state-machine protected object and a single activity task that controls the throttle. In state *Cruising*, the task repeatedly compares the current speed with the target speed and actuates on the throttle when needed. In state *Accelerating*, it maintains constant acceleration [21, 22]. This makes for smooth state changes. (If the driver's inputs do not create interrupts, an additional sampler task is needed.)

3 Event-thread patterns

The concurrent-activities and sequential-activities design/implementation patterns yield simple software solutions for many single-state-machine problems such as the garage door and the cruise controller. In more complex problems, we identify event threads and/or entities in the problem domain.

In the elevator-bank controller, for example, the life of an *elevator* entity can be shown in a state diagram. All cabins share a repository of out-standing requests for service, which is populated by sampling as travelers press buttons [18, 22]. Figure 3 shows a solution where each elevator task also has a separate repository for calls from its cabin buttons.

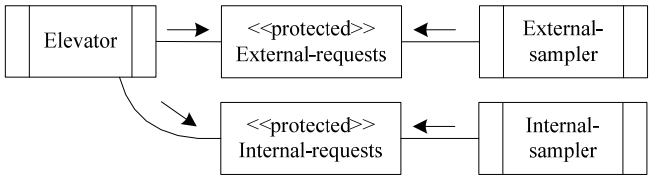


Figure 3 Communication diagram of the elevator software

This is an example of an **event-thread pattern,** a group of interacting entities, some of which enter data in a repository while others act on the data. The repository is consulted as a cabin approaches and leaves a floor. This pattern conforms in spirit to an architectural style called *repository* [10]. An event-thread pattern is a way to look at the problem and should not be confused with a design/implementation pattern, which deals with programming practicalities. Some even-thread patterns also map neatly onto the sequential-activities or concurrent-activities pattern, however

3.1 Event-thread patterns for resource sharing

Two event-thread patterns related to the important issue of domain-resource sharing have proven quite productive. If each resource user needs no more than one resource at a time, the event occurrences can be threaded either by resource or by resource user. ELM forces us to choose *one* view because otherwise, event occurrences involving a resource-user *and* a resource would belong to two event threads. In other words, there are two distinct *event-thread* patterns for resource sharing, the *resource-user-thread* pattern and the *resource-guard-thread* pattern as follows [22]:

A **resource-*user* event thread** is the life history of a resource-*user* entity in the problem domain and includes allocation events and events involving the shared resources. In the FMS problem for instance, *jobs* are resource-user entities. So are the *elevator* entities.

A resource-user event thread is typically implemented as a *sequential-activities* task, some of whose activities consist of waiting for resources. (This relationship is shown informally in Figure 1.) As the entities' surrogates, these tasks call operations on *monitor* and/or *semaphore* protected objects representing resources (Figure 1) and block until a resource becomes available.

Each **resource-*guard* event thread** represents the life history of a *resource entity* that services requests one by one. That leads to an implementation where resource-guard tasks are stations on a kind of assembly line where one task forwards an object representing the resource user to the next task. Queues implemented as protected objects connect the stations.

The two patterns are *dual* in that we can choose either a resource-guard-thread model or a resource-user-thread model of a given problem. No matter which model we think of first, it is a good mental habit always to work out its dual, which may turn out to be radically better [22].

Comparing different possible designs is essential in engineering [2, 9, 17, 23], and a discussion of alternatives considered should be part of the rationale for the solution ultimately chosen. Finding an alternative solution often takes imagination though, and we cannot even be sure that one exists [3]. But when the resource-sharing patterns apply, we can normally produce dual solutions.

4 Ex.: Flexible manufacturing system

The FMS problem illustrates problem-domain resource sharing. Flexible manufacturing differs from production lines in that each type of job visits its own series of workstations as defined by its process plan. In a particular FMS, multiple workstations of different types are connected by automated guided vehicles (AGVs) on a factory floor as shown in Figure 4 [6, 18, 19, 22]. Each workstation has an in-stand (I), an out-stand (O), and a robot arm (R) for moving the workpiece between an AGV, the stands, and the workstation tool.

Each job processes a single workpiece, which starts out and ends up in a storage bin in the automated storage and retrieval systems (ASRS). The workpiece is also staged in its bin when its next workstation is busy. From the bin, a forklift takes the piece to a storage stand where an AGV can get it. The piece then visits various workstations. When it cannot proceed directly from out-stand to in-stand, it is

staged in its storage bin. We shall first look at the sharing of workstations only; we return to the vehicles in 4.3

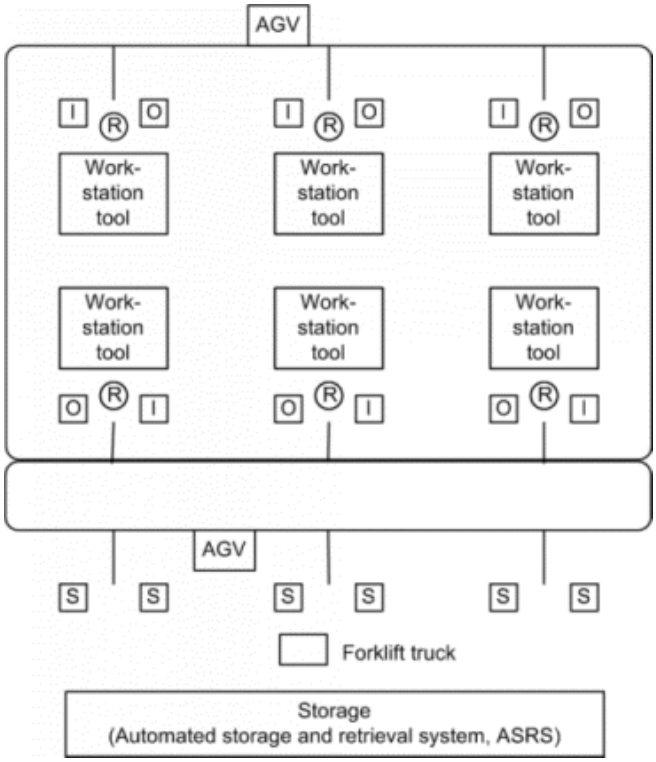


Figure 4 Layout of a simple FMS

4.1 Resource-*user* threads representing jobs

The *job* is a resource-user entity that acquires and releases exclusive control of workstations in competition with other jobs. Initially, the workpiece waits in its bin for access to the in-stand of a workstation of the right type. Once on the in-stand, it waits for the tool to become available. When done in the tool it moves onto the out-stand, if necessary after bumping the previous job off the stand. If completed or bumped, the job returns to its bin; otherwise it waits until the in-stand of its next workstation becomes available. The logic in pseudocode is as follows with assertions about the job's whereabouts in italics:

```
Get information about first job step
   while (job not completed)
         Workpiece is in bin
      Acquire workstation in-stand
      while (True)
         Workpiece is on storage stand or out-stand
         Travel to in-stand
         Acquire access to tool
         Wait for processing by tool
           Workpiece is being worked
         If out-stand busy, bump previous job
           Wait for it to clear stand
         Move to out-stand
         Get information about the next job step
         Break from inner loop if job completed
         Attempt to acquire in-stand for next step
         Wait until:   In-stand acquired
         or Bumping prompt: Break from inner loop
      Travel to storage bin
```

The job tasks are *sequential-activities* tasks with implicit state representation. It is appropriate to state-diagram the life of a job, but unlike the state machine in Figure 2, this one describes a single entity and has no co-occurring activities–the co-occurrence is instead between different jobs.

One difficulty inherent in the FMS problem is the situation where a job waits for either access to an in-stand or a bumping prompt, whichever comes first. A task representing the job can use an asynchronous select statement to block until either event occurs [6, 22].

4.2 Resource-*guard* threads for workstations

In a dual FMS solution, the jobs are passive, and each workstation has the following three resource-*guard* threads, collectively called *workstation threads*:

1. An *in-stand thread*, which finds an eligible job and moves the workpiece to the workstation. (The pseudocode is shown in 4.3.2.)
2. A *tool thread*, which waits for the workpiece to be done and moves it onto the out-stand.
3. An *out-stand thread*, which sends a workpiece to storage if it is done, or else waits for an in-stand or the bumping prompt. In case of bumping, the thread takes the workpiece to storage.

This is a valid event-thread model because every event occurrence (except job creation) is associated with a workstation. It is optimal because, at some instant, one workpiece can be on its way to the workstation, a second one can be done in the tool, and a third on its way to storage. Consequently, a situation–albeit unlikely–exists where all the threads have events occurring at the same time. This solution usually needs fewer tasks than the *job*-thread solution because jobs in storage have none.

4.3 Simultaneous exclusive access

When entities need exclusive access to *multiple* resources at once, resource-user threads are often the best solution. An implementation with implicit state representation exposes the resource acquisition and release clearly. Simultaneous exclusive access can set the stage for *deadlock*, a circular situation where–in the simplest case–each of two resource users waits indefinitely for a resource held by the other. In a system such as the FMS, the architect can use textbook deadlock prevention techniques to design the architecture to be deadlock free [22].

4.3.1 Deadlock prevention in the FMS example

The bumping in the FMS prevents deadlock by eliminating indefinite waits. Without bumping, two workstations could each have a workpiece on its in-stand and out-stand and in its tool, and each workpiece on an out-stand could be waiting for the other in-stand. Bumping stops jobs from keeping an out-stand indefinitely and thereby prevents deadlock.

Besides workstations, FMS jobs need transportation, which requires exclusive access to multiple resources at once,

such as an in-stand, a storage stand, and a forklift. We can eliminate circularity at this level by stipulating that each job acquire resources in a given order called a *locking order*. This is a *partial order* where $R < S$ means that if a job needs exclusive access to resources R and S at once, R must be acquired before S. The order *in-stand* < *storage stand* < *forklift*, *storage stand* < *AGV*, and *in-stand* < *AGV* works for the FMS. Here is a more complete pseudocode for the *job* task type with the locking order observed:

```
Get information about first job step
while (job not completed)
        Workpiece is in bin
   Acquire workstation in-stand
   Acquire storage stand
   Forklift.To_stand;
   while (True)
        Workpiece is on storage stand or out-stand
      Acquire AGV; Travel by AGV to in-stand
      Release AGV, and storage stand or out-stand
      Wait for processing by tool
        Workpiece is being worked
      If out-stand busy, bump previous job, wait for it to clear
         out-stand
      Move to out-stand
      Get information about the next job step
      Break from inner loop if job completed
      Attempt to acquire in-stand of workstation for next step
      Wait until: In-stand acquired
      or   Bumping prompt: Break from inner loop
   Acquire storage stand
   Acquire AGV; Travel to storage stand; Release AGV
   Forklift.To_bin;
   Release storage stand
```

Semaphore protected objects (Figure 1) guard the workstations and AGVs. The *Acquire* and *Release* calls are in-line, making it plain to see which resources are held and enforce the locking order. The *monitor* protected object *Forklift* encapsulates entire moves in the operations *To_stand* and *To_bin*. (The AGVs could be handled similarly.)

4.3.2 Solution with workstation threads

A change from *job* tasks to workstation tasks barely affects the protected objects that control access to forklifts and AGVs. Those objects assume only that each calling task represents a job. This obviously holds with *job* tasks, and it holds with workstation tasks too: They acquire AGVs, etc., on jobs' behalf. While the tasks are resource *guards* with respect to the workstations, they are resource-*user* tasks with respect to vehicles. As an example, here is the *in-stand* task in pseudocode:

```
while (True)
   Find eligible job
   if job in storage then
      Acquire storage stand
      Forklift.To_stand
  Acquire AGV; Take workpiece to in-stand;
  Release AGV and storage/out-stand
```

The next section discusses task architectures generally and uses the FMS example for illustration.

5 Tasks as architectural elements

Because the term "architecture" tends to make us think of buildings, "software architecture" may conjure up an image of a static module structure. But unlike a building, a reactive software system is a machine with dynamic characteristics [23]. Tasks capture those dynamics. They are not modules but still loosely coupled building blocks.

ELM tasks can be thought about in problem-domain terms. For example, the form of each FMS architecture shows how it manages the real-world problem outside the software. That way, the task architecture can expose important problem aspects much as the form given to physical device can bring out its function.

Because task architecture defines the machinery of interacting tasks, it allows us to create early running prototypes with simplified logic to study performance. Additional scaffolding tasks can simulate the sources of external impulses and also respond to actions taken by the software. For example, a task that simulates an AGV can interact with *job* tasks or workstation tasks. Such scaffolding tasks can often be sequential-activities tasks with implicit state representation [18].

5.1 Conceptual integrity and key ideas

Software architecture is said to have *conceptual integrity* if all its parts serve one central purpose. Such architecture appears to flow from a single mind or perhaps two minds, "acting *uno animo*" [3, 4]. Conceptual integrity helps to make the architecture an "intellectually graspable model" and sometimes a "reusable and transferable abstraction" [2]. The integrity can often be manifest in a *key idea*.

Conceptual integrity is perhaps nowhere more important than in task architectures. To this end, ELM prescribes certain styles, which can be significant also in what they exclude [23]. For example, an FMS architecture can have *job* tasks or workstation tasks–not both. Thus, within the ELM confines, an idea such as "a task per job" determines an architecture and distinguishes it from possible others.

A simple key idea can carry much meaning because ELM architectures usually contain far fewer task types than there are classes in an object-oriented design or transforms in a dataflow model. Scale is achieved by instantiation, not by adding element types [17]. By twisting and turning an idea in our minds, we may come up with a better one; just figuring out the dual of a resource-sharing solution may open new, surprising vistas on a problem.

If the key idea is clear enough, someone not involved in the original design effort can build on it. The FMS solution where each job pursues its completion came from students at the Wang Institute [18, 22]. Later, Rob Scott proposed that instead, *workstations* procuring jobs to work on could be seen as the driving forces. That idea together with ELM's principles let us visualize such a solution and validate it as a consistent and worthwhile alternative.

6 Conclusion

An abundance of computing power has created new conditions for the design of reactive software. History shows that new tools and materials ultimately make us design to their strengths. Factories once replaced central steam engines with central electric motors and thereby got the coal off the premises. But they made a quantum leap only when all over the plant, power lines to smaller motors replaced all the belts and pulleys [5].

So it is with tasking. ELM exploits the new hardware to free tasking from practical constraints and base it instead on concurrency inherent in the problem. In the same manner, programming in general has evolved away from the computer's physical structure and limitations [11]. Besides, the idea of arranging happenings in the problem domain into event threads also turns out to be quite teachable.

We should make software architecture more understandable to stakeholders somewhat removed from the "code". To reach that goal, we should strive to make software as transparent as clockwork. "[S]imple designs can be communicated easily; complex designs are hard to explain" [24]. A task architecture can reveal the software mechanics without much technical detail. Simple designs can boost correctness too, for the easier it is to study and discuss an architecture, the more flaws and inconsistencies we find. And having fully understood the architecture, a maintainer can repair and extend it in the style of the original without fear of upsetting some arcane, delicate balance.

Acknowledgments

Many reviewers and in particular those associated with Ada-Europe provided highly relevant and much appreciated feedback.

References

[1] G. R. Andrews (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison-Wesley Longman.

[2] L. Bass, P. Clements, and R. Kazman (2003), Software Architecture in Practice, 2nd Ed. Addison-Wesley.

[3] F. P. Brooks, Jr. (1995), The Mythical Man-Month, Anniversary Ed., Addison-Wesley.

[4] F. P. Brooks, Jr. (2010), *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley.

[5] E. Brynjolfsson, P. Hofmann, and J. Jordan (2010). *Cloud Computing and Electricity: Beyond the Utility Model*, CACM, vol 53 no 5, pp 32-34.

[6] J. R. Carter, and B. I. Sandén (1998), *Practical Uses of Ada-95 Concurrency Features*, IEEE Concurrency, vol 6 no 4, pp 47-56.

[7] B. P. Douglass (2009), *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*, Addison-Wesley Professional.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[9] D. Garlan and D. Perry (1995), *Introduction to the Special Issue on Software Architecture*, IEEE Transactions on Software Engineering, vol 23 no 4, pp 269-274.

[10] D. Garlan and M. Shaw (1996), *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall.

[11] D. H. Gelernter (1998), *Machine Beauty: Elegance and the Heart of Technology*, Basic Books.

[12] D. Harel and A. Pnueli (1995), *On the Development of Reactive Systems*, in *Logics and Models of Concurrent Systems*, K. R. Apt (ed.), Springer, 477-498.

[13] M. H. Klein, T. Ralya, W. Pollak, R. Obenza, and M. Gonzalez-Harbour (1993), *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers.

[14] Ph. Kruchten (1955), *The 4+1 Model of Architecture*, IEEE Software, vol 12 no 6, pp 42-50.

[15] E. A. Lee (2006), *The Problem with Threads*, IEEE Computer, vol 39 no 5, pp 33-42.

[16] E. A. Lee (2009), *Computing Needs Time*, CACM, vol 52 no 5, pp 70-79.

[17] D. E. Perry and A. L. Wolf (1992), *Foundations for the Study of Software Architecture*, ACM Software Engineering Notes vol 17 no 4, pp 40-52.

[18] B. I. Sandén (1994), *Software Systems Construction with examples in Ada*, Prentice-Hall.

[19] B. I. Sandén (1997), *Modeling Concurrent Software*, IEEE Software, vol 14 no 5, pp 93-100.

[20] B. I. Sandén (2003). *Entity Life Modeling: Modeling a Thread Architecture on the Problem Environment*, IEEE Software, vol 20 no 3, pp 70-78.

[21] B. I. Sandén and J. Zalewski. Designing state-based systems with entity-life modeling. Journal of Systems and Software, 79:1 (Jan. 2006), 69-78.

[22] B. I. Sandén (2011), *Design of Multithreaded Software: The Entity-Life Modeling Approach*, IEEE Computer Society Press/Wiley.

[23] R. N. Taylor, N. Medvidovic, and E. M. Dashofy (2009), *Software Architecture: Foundations, Theory, and Practice*, Wiley.

[24] J. Waldo (2006), *On System Design*. Proc. OOPSLA'06, October 22-26, Portland, OR, pp 467-479.

[25] R. J. Wieringa (2003), *Design Methods for Reactive Systems: Yourdon, Statemate, and the UML*, Morgan Kauffman.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/category/developers-center/gems/.

Gem #113: Visitor Pattern in Ada

Emmanuel Briot, AdaCore

Date: 07 November 2011

Abstract: The visitor pattern is a design pattern that provides a way to execute specific methods on an object (the visitor) depending on the type of another object. Since the exact subprogram called depends on both types of the objects, this pattern is often called double dispatching.

Let's get started…

Imagine that you have a UML model and you want to generate code from it. A convenient approach is to have a "code generator" object, which has a set of subprograms to handle each kind of UML element (one that generates code for a class, one that generates code for an operation, etc.).

One way to implement this is by using a big series of if statements, of the form if Obj in CClass'Class then, which is rather inelegant and inefficient.

Another approach is to use discriminated types. A case statement on the discriminant is then efficient, and Ada will check that all discriminant values are covered. The problem is that then you would need to use case statements for all clients of the types in your application. Here, we prefer to use tagged types, to take advantage of Ada's OOP capabilities, so the case statement cannot be used.

Let's consider a specific example. Again, taking the UML example, assume we have the following types. These are only very roughly similar to the actual UML metamodel, but will be sufficient for our purposes. In practice, these types would be automatically generated from the description of the UML metamodel.

```
type NamedElement is tagged private;
type CClass is new NamedElement with private;
type PPackage is new NamedElement with private;
```

In addition, a visitor class is declared, which will be overridden by the user code, for instance, to provide a code generator, a model checker, and so on:

```
type Visitor is abstract tagged null record;

procedure Visit_NamedElement
   (Self : in out Visitor; Obj : NamedElement'Class) is null;
-- No parent type, do nothing

procedure Visit_CClass (Self : in out Visitor;
                        Obj : CClass'Class) is
begin
   -- In UML, a "Class" inherits from a "NamedElement".
   -- Concrete implementations of the visitor might want
   -- to work at the  "NamedElement" level (so that their
   -- code applies to both a Class and a Package,
   -- for instance), rather than duplicate the work for each
   --  child of NamedElement. The default implementation
   --  here is to call the parent type's operation.

   Self.Visit_NamedElement (Obj);
end Visit_Class;

procedure Visit_PPackage (Self : in out Visitor;
                          Obj : PPackage'Class) is
begin
   Self.Visit_NamedElement (Obj);
end Visit_PPackage;
```

We then need to add one primitive Visit operation to each of the types created from the UML metamodel:

```
procedure Visit (Self : NamedElement;
                 V : in out Visitor'Class) is
begin
   --  First dispatching was on "Self" (done by the
   -- compiler).
   --  Second dispatching is simulated here by calling the
   -- right primitive operation of V.

   V.Visit_NamedElement (Self);
end Visit;

overriding procedure Visit (Self : CClass;
                            V : in out Visitor'Class) is
begin
   V.Visit_CClass (Self);
end Visit;

overriding procedure Visit (Self : PPackage;
                            V : in out Visitor'Class) is
begin
   V.Visit_PPackage (Self);
end Visit;
```

All of the code described above is completely systematic, and as such could and should be generated automatically as much as possible. The "Visit" primitive operations should never be overridden in user code in the usual case. On the other hand, the "Visit_…" primitives of the visitor itself should be overridden when it makes sense. The default implementation is provided just so the user has the choice at which level do to the overriding.

Now let's see what a code generator would look like. We'll assume that we are only interested, initially, in doing code generation for classes. Other types of elements (such as operations) will call the default implementation for their

visitor (Visit_Operation, for instance), which then calls the visitor for its parent (Visit_NamedElement) and so on, until we end up calling a Visit operation with a null body. So nothing happens for those, and we don't need to deal with them explicitly.

The code would be something like the following:

```
type CodeGen is new Visitor with private;

overriding procedure Visit_CClass
  (Self : in out Codegen; Obj : CClass'Class) is
begin
   ...; -- Do some code generation
end Visit_CClass;

procedure Main is
  Gen : CodeGen;
begin
  for Element in All_Model_Elements loop
    -- Pseudo code
    Element.Visit (Gen);  -- Double dispatching
  end loop;
end Main;
```

If we wanted to do model checking, we would create a type Model_Checker, derived from Visitor, that overrides some of the Visit_* operations. The body of Main would not change, except for the type of Gen.

When using this in practice, there are a few issues to resolve. For instance, the UML types need access to the Visitor type (because it appears as a parameter in their operations). But a visitor also needs to see the UML types for the same reason. One possibility is to put all the types in the same package. Another is to use "limited with" to give visibility on access types, and then pass an access to Visitor'Class as a parameter to Visit.

Here is a full example. This example must be compiled with the "-gnat05" switch since it uses Ada 2005 features such as the limited with clause and prefixed call notation.

```
with UML;       use UML;
with Visitors;  use Visitors;
with Ada.Text_IO; use Ada.Text_IO;

procedure Main is
  type Code_Generator is new Visitor with null record;

  overriding procedure Visit_CClass
    (Self : in out Code_Generator;
     Obj : in out CClass'Class) is
  begin
    Put_Line ("Visiting CClass");
  end Visit_CClass;

  Tmp1 : NamedElement;
  Tmp2 : CClass;
  Tmp3 : PPackage;

  Gen  : aliased Code_Generator;

begin
  Tmp1.Visit (Gen'Access);  -- No output
  Tmp2.Visit (Gen'Access);  -- Outputs "Visiting CClass"
  Tmp3.Visit (Gen'Access);  -- No output
end Main;

limited with Visitors;
package UML is
  type NamedElement is tagged null record;
  procedure Visit
    (Self       : in out NamedElement;
     The_Visitor : access Visitors.Visitor'Class);

  type CClass is new NamedElement with null record;
  overriding procedure Visit
    (Self       : in out CClass;
     The_Visitor : access Visitors.Visitor'Class);

  type PPackage is new NamedElement with null record;
  overriding procedure Visit
    (Self       : in out PPackage;
     The_Visitor : access Visitors.Visitor'Class);
end UML;

with Visitors;  use Visitors;
package body UML is

  procedure Visit
    (Self       : in out NamedElement;
     The_Visitor : access Visitors.Visitor'Class) is
  begin
    The_Visitor.Visit_NamedElement (Self);
  end Visit;

  overriding procedure Visit
    (Self       : in out CClass;
     The_Visitor : access Visitors.Visitor'Class) is
  begin
    The_Visitor.Visit_CClass (Self);
  end Visit;

  overriding procedure Visit
    (Self       : in out PPackage;
     The_Visitor : access Visitors.Visitor'Class) is
  begin
    The_Visitor.Visit_PPackage (Self);
  end Visit;

end UML;

with UML;  use UML;

package Visitors is
  type Visitor is abstract tagged null record;

  procedure Visit_NamedElement
    (Self : in out Visitor; Obj : in out NamedElement'Class);
  procedure Visit_CClass
    (Self : in out Visitor; Obj : in out CClass'Class);
  procedure Visit_PPackage
    (Self : in out Visitor; Obj : in out PPackage'Class);
end Visitors;
```

```
package body Visitors is

   procedure Visit_NamedElement
     (Self : in out Visitor;
      Obj : in out NamedElement'Class) is
   begin
     null;
   end Visit_NamedElement;

   procedure Visit_CClass
     (Self : in out Visitor; Obj : in out CClass'Class) is
   begin
     Self.Visit_NamedElement (Obj);
   end Visit_CClass;

   procedure Visit_PPackage
     (Self : in out Visitor; Obj : in out PPackage'Class) is
   begin
     Self.Visit_NamedElement (Obj);
   end Visit_PPackage;

end Visitors;
```

Gem #117: Design Pattern: Overridable Class Attributes in Ada 2012

Emmanuel Briot, of AdaCore

Date: 30 January 2012

Abstract: In this Gem we consider how to realize the capability of "class attributes" (such as supported in Python) using Ada.

Let's get started…

Most object-oriented programming languages provide a facility for declaring variables that are shared by all objects of a given class. In C++, these are called "static members" (and use the "static" keyword), and similarly Python has the notion of "class attributes".

Let's consider an example where this is useful. For instance, let's say we want to define the notion of a block of text that is generated by expanding a template (perhaps after we replace some parameters in that template, as can be done with AWS's templates parser, for instance). Once we have computed those parameters, we might want to generate multiple outputs (for instance HTML and CSV). Only the template needs to change, not the computation of the parameters.

Typically, such as in Python, the template could be implemented as a class attribute of the Text_Block class. We can then create templates that need the same information but have a different output simply by extending that class:

```
class Text_Block(object):
   template = "somefile.txt"
   def render (self):
      # ... compute some parameters
      # Then do template expansion
      print "processing %s" % self.__class__.template

class Html_Block(Text_Block):
   template = "otherfile.html"
```

In this example, we chose to use a class attribute rather than the usual instance attribute (self.template). This example comes from the implementation of GnatTracker: in the web server we create a new instance of Text_Block for every request we have to serve. For this, we use a registry that maps the URL to the class we need to create. It is thus easier to create a new instance without specifying the template name as a parameter, which would be required if the template name was stored in the instance. Another reason (though not really applicable here) is to save memory, which would be important in cases where there are thousands of instances of the class.

Of course, the approach proposed in this Gem is not the only way to solve the basic problem, but it serves as a nice example of one of the new Ada 2012 features.

C++, like Ada, does not provide a way to override a static class member, so it would use a similar solution as described below.

Since Ada has no notion of an overridable class attribute, we'll model it using a subprogram instead (the only way to get dispatching in Ada). The important point here is that we want to be able to override the template name in child classes, so we cannot use a simple constant in the package spec or body.

```
type Text_Block is tagged null record;
function Template (Self : Text_Block) return String;
function Render (Self : Text_Block) return String;

function Template (Self : Text_Block) return String is
   pragma Unreferenced (Self);
begin
   return "file_name.txt";
end Template;
```

The parameter Self is only used for dispatching (so that children of Text_Block can override this function). Since we prefer to compile with "-gnatwu" to get a warning on unused entities, we indicate to the compiler that it is expected that Self is unreferenced.

We could make the function Template inlinable, which might be useful in a few cases (for instance if called from Render in a nondispatching call), but in general there will be no benefit because Template will be a dispatching call, which requires an indirect call and thus wouldn't benefit from inlining.

And that's it. We have the Ada equivalent of a Python class member.

But so far there is nothing new here, and this approach is rather heavy to write. For instance, the body of Render could contain code like:

```
pragma Ada95;

function Render (Self : Text_Block) return String is
   T : constant String :=
       Template (Text_Block'Class (Self));
begin
   .. prepare the parameters for template expansion
   .. substitute in the template and return it
end Render;
```

Fortunately, Ada 2012 provides an easier way to write this, using the new feature of expression functions. Since Template is a function that returns a constant, we can declare that directly in the spec, and remove the body altogether. The spec will thus look like:

```
pragma Ada_2012;

type Text_Block is tagged null record;
function Template (Self : Text_Block) return String
  is ("filename.txt");
function Render (Self : Text_Block) return String;
```

This is a much lighter syntax, and much closer to how one would do it in Python (except we use a function instead of a variable to represent a class member). A child of Text_Block would override Template using the same notation:

```
type Html_Block is new Text_Block with null record;
overriding function Template (Self : Html_Block)
  return String is ("otherfile.html");
```

Compared to Python, this is in fact more powerful, because some of the children could provide a more complex body for Template, so we are not limited to using the value of a simple variable as in Python. In fact, we can do this in the spec itself, by using a conditional expression (another new feature of Ada 2012):

```
pragma Ada_2012;

type Text_Block is tagged null record;
function Template (Self : Text_Block) return String
   is (if Self.Blah then "filename.html" else "file2.json");
function Render (Self : Text_Block) return String;
```

Finally, we can also make the body of Render slightly more familiar (in terms of object-oriented notation) using the dotted notation introduced in Ada 2005:

```
function Render (Self : Text_Block) return String is
  T : constant String := Text_Block'Class (Self).Template;
begin
  .. prepare the parameters for template expansion
  .. substitute in the template and return it
end Render;
```

Now the call to Template looks closer to how it would appear in those languages that provide overridable class members. Some will argue that this doesn't look like a function call and thus is less readable, since we don't know that we are calling a function. This is a matter of taste, but at least we have the choice.

There is one thing we have lost, temporarily, in the declaration of Template. If we compile with -gnatwu, the compiler will complain that Self is unreferenced. There is currently no way to add a pragma Unreferenced within an expression function. This has generated a discussion here at AdaCore and the issue is not resolved yet. The current two proposals are either to always omit the unused parameter warning when a function has a single parameter and it controls dispatching (precisely to facilitate this class member pattern), or else to use an Ada 2012 aspect for this, as in the following:

```
function Template (Self : Text_Block) return String
  is ("filename.html")
with Unreferenced => Self;
```

Note also that the use of expression functions in this Gem requires a very recent version of GNAT: the expression function feature wasn't available in older versions, and the initial implementation had some limitations.

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