ADA USER JOURNAL

Volume 34
Number 2
June 2013

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. The language of the journal is English.

Although the title of the Journal refers to the Ada language, related topics, such as reliable software technologies, are welcome. More information on the scope of the Journal is available on its website at *www.ada-europe.org/auj*.

The Journal publishes the following types of material:

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

Further details on our approach to these are given below. More complete information is available in the website at *www.ada-europe.org/auj*.

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues

The *Ada User Journal* is open to consider the publication of proceedings of workshops or panels related to the Journal's aims and scope, as well as Special Issues on relevant topics.

Interested proponents are invited to contact the Editor-in-Chief.

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. Our readers need not surf the web or news groups to find out what is going on in the Ada world and in the neighbouring and/or competing communities. We will 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 electronically. Authors are invited to contact the Editor-in-Chief by electronic mail to determine the best format for submission. The language of the journal is English.

Our refereeing process aims to be rapid. Currently, accepted papers submitted electronically are typically published 3-6 months after submission. Items of topical interest will normally appear in the next edition. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

The June issue of the Ada User Journal is finalised shortly after the Ada-Europe 2013 conference, which took place in Berlin, Germany, in the week of June 10 to 14. The organizers must be congratulated for a successful conference, with a rich program, and a pleasant social and networking atmosphere. As announced during the conference, next year the Ada-Europe conference will take place in Paris, France, in the week of 23-27 June, 2014. A great opportunity for Ada and Reliable Software practitioners and enthusiasts to present their work and for the community to connect in an enjoyable scenario. You can find the preliminary call for papers in the Forthcoming Events section of this issue. Note that the program of the conference results from the contributions of the community, by means of the submission of papers, presentation, tutorials or workshops. I would like to both encourage, and insist in asking for, your contribution!

Also in the Events section, the Journal provides the announcement and highlights of SIGAda HILT 2013, which will take place this year in Pittsburgh, USA, in the week of November 10-14. As usual the News Digest and Calendar sections, prepared by the respective Editors, Jacob Sparre Andersen and Dirk Craeynest, complete the first part of the issue.

As for the technical contents, the issue provides another chapter of the Ada 2012 Rationale, which is being written by John Barnes. This chapter concludes the presentation of the predefined library, with the description of the improvements to containers, also concluding the core chapters of the Rationale. In the September issue of the Journal we will publish the Ada 2012 Rationale epilogue.

The reader may remember a previous announcement and mention of the International Real-Time Ada Workshop (IRTAW 2013). The Journal will publish a report on this important event in the September issue, and there are plans to also publish this year the summaries of the workshop sessions. In the meanwhile, we consider important to provide the readers with the session summaries of the previous IRTAW (2011), which are being published this year. It is possible to note that many issues which were open in 2011 are increasingly relevant, and so it is not a surprise that they were part of this year's workshop program.

The issue continues with a short summary about the one day Ada Conference UK 2013, that took place April 25 in Birmingham, UK. To finalise, the Ada Gems section provides a gem on Characters and Encoding Schemes, by Emmanuel Briot and the series of gems on Su(per)btypes in Ada 2012, by Yannick Moy, both with AdaCore, France.

Luís Miguel Pinho
Porto
June 2013
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

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

Contents

Ada-related Events

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

Ada-France at Solutions Linux

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Mon, 18 Mar 2013 11:43:45 +0100
Subject: Ada-France au salon Solutions Linux
To: liste ada-france <ada-france@ada-france.org>

Comme l'anné dernière, Ada-France aura son stand dans le village associatif, au salon Solutions Linux qui aura lieu au CNIT La Défense les 28 et 29 mai.

Toutes les bonnes volontés sont les bienvenues! Par exemple:

- Venir staffer le stand quelques heures (pas obligatoirement - mais bienvenu - toute la journée). C'est sympa, on rencontre des gens, et on leur fait découvrir qu'Ada, ce n'est pas si ringard que ça..
- Envoyer des jolies applications, démos, etc, de préference attractives et spectaculaires.
- Des posters, ou simplement de bonnes idées pour faire la promo d'Ada…

[Ada-France will be present at Solutions Linux in Paris, May 28th and 29th. —sparre].

Ada and Education

Railway simulators

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue May 28 11:40:32 CEST 2013
Subject: Railway simulators

Looking through some of the Ada repositories on GitHub [1], I noticed two repositories related by their subjects:

- Railway simulators
- Course-work

The repositories are:

- morambro/TrainProject [2]
- rostgaard/railway-validator [3]

It appears that (at least some) students at both the Technical University of Denmark and the University of Padova get to program in Ada.

[1] https://github.com/search?q=language%3AAda&type=Repositories

[2] https://github.com/morambro/TrainProject

[3] https://github.com/rostgaard/railway-validator

Ada-related Resources

Safe, dynamic task creation

From: R. Toyler Croy / agentdero
Date: Sat, 9 Mar 2013
Subject: Safe, Dynamic Task Creation in Ada
URL: http://unethicalblogger.com/2013/03/09/dynamic-tasks-in-ada.html

A few years ago, Ada become my hobby/tinker programming language of choice, for a number of reasons, concurrency being one of them. In this post I'd like to walk you through an example of dynamic task creation in Ada, which uses Ada.Task_Termination handlers, a new feature in Ada 2005.

[…]

[A nice example of how to use Ada.Task_Termination. —sparre].

Social Media Sites

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed Apr 17 2013
Subject: Highlighting Ada on Social Media Sites
URL: http://www.adaic.org/2013/04/highlighting-ada-on-social-media-sites/

Based on a discussion on comp.lang.ada, we've created a new category on our "Learning Materials" for social media sites, and added links to the Ada groups on a number of them:

- Stackoverflow – Questions and answers for programmers [1]
- Google+ – Ada Programming Community [2]
- LinkedIn – Ada Group [3]

All of these provide places to get help and information about Ada from other Ada users. We also added Planet Ada [4], a news and information site, to the website listings. These join the Reddit [5] and Wikibooks [6] sites, along with the old stand-by of the comp.lang.ada newsgroup, as places to go for Ada information.

[1] http://stackoverflow.com/questions/tagged/ada

[2] https://plus.google.com/u/0/communities/102688015980369378804

[3] http://www.linkedin.com/groups?gid=114211

[4] http://planet.ada.cx/

[5] http://www.reddit.com/r/ada/

[6] http://en.wikibooks.org/wiki/Ada_Programming

Updated Rationale available

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu May 16 2013
Subject: Updated Ada 2012 Rationale available
URL: http://www.adaic.org/2013/05/updated-ada-2012-rationale-available/

An updated edition of the Ada 2012 Rationale is available at:

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

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

The Rationale for Ada 2012 provides an overview of new Ada 2012 features,

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

Repositories of open source software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue May 28 12:07:52 CEST 2013
Subject: Repositories of Open Source software

- AdaForge: 7 repositories [1]
- Bitbucket: 48+ repositories [2,3]
- Codelabs: 10+ repositories [4]
- GitHub: 344 repositories [5]
  91 developers [6]
- Rosetta Code: 569 examples [7]
  25 developers [8]
- Sourceforge: 219 repositories [9]

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

[2] https://bitbucket.org/repo/all/relevance?name=binding&language=ada

[3] https://bitbucket.org/repo/all/relevance?name=ada&language=ada

[4] http://git.codelabs.ch/

[5] https://github.com/search?q=language%3AAda&type=Repositories

[6] https://github.com/search?q=language%3AAda&type=Users

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

[8] http://rosettacode.org/wiki/Category:Ada_User

[9] http://sourceforge.net/directory/language%3Aada/

[An update to my overview in issue 33-4. —sparre]

The #58 most popular language on GitHub

From: GitHub
Date: Tue May 28 10:24:36 CEST 2013
Subject: Ada is the #58 most popular language on GitHub
URL: https://github.com/languages/Ada

Most starred this month:

- AdaDoom3 / AdaDoom3 [1]
- cforler / Ada-Crypto-Library [2]
- rtyler / ada-playground [3]
- flyx86 / OpenGLAda [4]
- darkestkhan / lazyfoo [5]

Most starred overall:

- Lucretia / tamp [6]
- AdaDoom3 / AdaDoom3 [1]
- karakalo / old-lumen [7]
- cforler / Ada-Crypto-Library [2]
- ThomasLocke / yolk [8]

Most forked this month:

- AdaDoom3 / AdaDoom3 [1]
- Calvin-he / docclustering [9]
- rtyler / tinywm-ada [10]
- dsanson / Words [11]

Most forked overall:

- AdaDoom3 / AdaDoom3 [1]
- rtyler / tinywm-ada [10]
- AdaHeads / Alice [12]
- Lucretia / tamp [6]
- persan / zeromq-Ada [13]

[91 Ada developers are registered on GitHub [14] and GitHub knows of 344 Ada projects [15]. —sparre]

References:

[1] https://github.com/AdaDoom3/AdaDoom3

[2] https://github.com/cforler/Ada-Crypto-Library

[3] https://github.com/rtyler/ada-playground

[4] https://github.com/flyx86/OpenGLAda

[5] https://github.com/darkestkhan/lazyfoo

[6] https://github.com/Lucretia/tamp

[7] https://github.com/karakalo/old-lumen

[8] https://github.com/ThomasLocke/yolk

[9] https://github.com/Calvin-he/docclustering

[10] https://github.com/rtyler/tinywm-ada

[11] https://github.com/dsanson/Words

[12] https://github.com/AdaHeads/Alice

[13] https://github.com/persan/zeromq-Ada

[14] https://github.com/search?q=language%3AAda&type=Users&s=repositories

[15] https://github.com/search?q=language%3AAda&type=Repositories&s=updated

Ada-related Tools

Orto – more than just a command line parser

From: Björn Persson
Date: Fri, 8 Mar 2013 10:02:07 +0100
Subject: Orto – more than just a command line parser
URL: http://adacl.sourceforge.net/pmwiki.php/Main/Orto

The problem

In principle, command line parameters of a program work the same way as parameters of a subprogram, but there is a huge difference in the amount of work required of the programmer. In a subprogram you declare the parameters with name and type, and to use the value of a parameter you just write its name. The compiler does the rest. Command line parameters, on the other hand, are just a list of strings. Before the program can start its work it has to analyze the command line carefully. It has to check that all the required parameters are present, that there aren't any unrecognized parameters, and that there aren't multiple instances of parameters that are only meaningful to give once. It must interpret the parameters as values of different types and check that the values are within their constraints. If anything is wrong it has to print an informative error message.

Writing code to do all of this in each program can be quite tedious, and it is tempting to do it the easiest way possible. This easily makes the command line syntax unnecessarily strict or difficult to learn, making the program harder to use.

Orto to the rescue

Orto takes care of most of this work and makes command line parameters almost as easy to work with as parameters of subprograms. You declare your command line parameters with name and type, specify whether they are mandatory or optional and whether they may occur more than once, and maybe provide a default value. Orto will then analyze the command line, identify the parameters and interpret them according to their specified types. It checks that the command line is in every way correct, and prints error messages if any errors are found. If all is correct you can then retrieve the value of any parameter with a simple function call, and it is delivered as a value of its specified type, ready to use without further translation.

For numeric parameters you can define a unit that the value is measured in. Orto will then understand and handle the standard unit prefixes. If the unit is "m" for meters, the user may type "5km" or "98cm", and Orto will recognize the prefix and multiply the value by the corresponding factor.

Orto also prints help texts and version numbers. The same parameter definitions that rule how Orto interprets the parameters are also used to generate the help text, so there is no risk of discrepancies between the help text and how the program actually interprets the command line.

Orto is therefore far more than a command line parser; it's a complete command line parameter handler.

PragmAda reusable components

From: Jeffrey R. Carter <pragmada@pragmada.x10hosting.com>
Date: Mon, 11 Mar 2013 12:48:37 -0700
Subject: New Version of the PragmAda Reusable Components Available
Newsgroups: comp.lang.ada

There is a new version of the PragmAda Reusable Components (PragmARCs) available at

http://pragmada.x10hosting.com/pragmarc.htm

This version adds a parallel version of quick sort to the components. Unit PragmARC.Sort_Quick_In_Place has changed from a procedure to a package exporting two procedures; programs using an older version of Sort_Quick_In_Place will need to be modified to use the new version.

There is also now available a beta version of the components suitable for compilation with a compiler that implements Ada-95 as amended by Amendment 1 (2007). There are a number of differences between this version and the Ada-95 version of the components, most noticeably in the implementation of the data structures. This version is not backward compatible with the Ada-95 version.

As always, error reports, comments, and suggestions are welcome from all users.

Simple components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 16 Mar 2013 09:46:17 +0100
Subject: ANN: Simple components v3.22 multiple connections servers support
Newsgroups: comp.lang.ada

This version provides support for designing multiple connections TCP/IP servers. The connections are handled by one task. Although servers backed by a pool of working tasks are supported too. The server uses socket-select for socket I/O (based on GNAT.Sockets).

Since programming a state machine parsing incoming packets is difficult, the library eases that. You simply put types representing elements of the packet into the custom connection type like this:

```
type My_Protocol is new
        State_Machine is record
   Header : Big_Endian.Unsigneds.
                    Usigned_16_Data_Item;
   Value : Big_Endian.Unsigneds.
                    Usigned_32_Data_Item;
   …
end record;
overriding procedure Process_Packet
        (Client : in out My_Protocol);
```

The state machine will notify when all items are received by calling Process_Packet. The library provides implementations of big- and little-endian encoded integers, unsigneds, IEEE floats, as well as null terminated strings and equivalent of variant records.

A sample implementation of a fully functional HTTP implementation is included. It does not access host file system, it does not allocate memory dynamically on receipt unless at user request. CGI and multipart bodies are supported.

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

[See also Ada User Journal volume 33 issue 4. —sparre]

From: Mário Alves
Date: Sat, 16 Mar 2013
Subject: Simple components v3.22 supports multiple connections servers
URL: http://www.linkedin.com/

This is very good news.

The HTTP part is competing with Adacore's AWS. A problem with AWS is the difficulty to install and to build applications with it. Requires makefiles and GNAT projects. I have used DK's Simple Components library in the past and found it much easier to install and build with it.

Displaying characters on MOD-LCD3310 by using Olimexino-328

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon, 18 Mar 2013
Subject: Displaying Characters on MOD-LCD3310 by using Olimexino-328 with Ada
URL: http://arduino.ada-language.com/displaying-characters-on-mod-lcd3310-by-using-olimexino-328-with-ada.html

MOD-LCD3310 is Nokia 3310 display clone from Olimex. It provides black and white screen with 84x48 resolution, which is a great choice if 7-segment display or 2x16 LCD isn't enough.

MOD-LCD3310 uses UEXT connector found from almost every Olimex board. In case you don't have one, you can follow the UEXT specification and just connect the pins.

Like how I have done with Olinuxino-imx233 micro board:

[As usual, Tero shows us how we can have fun with hardware and Ada. —sparre]

Ada 95 Booch components

From: Simon Wright <simon@pushface.org>
Date: Fri, 22 Mar 2013 16:28:41 +0000
Subject: ANN: Ada 95 Booch Components release 20130322
Newsgroups: comp.lang.ada

This release is available at Sourceforge [1].

Minor changes only:

- Added BC.Indefinite_Unmanaged_Containers.Queues.Ordered.
- A problem with building any indefinite dynamic form under GCC 4.7 has been resolved.
- BC.Support.Indefinite_Dynamic was missing.
- Includes a patch to allow use of bc.gpr on Debian 6.

[1] https://sourceforge.net/projects/booch95/files/booch95/20130322/

AVR-Ada

From: Rolf Ebert <rolf.ebert.gcc@gmx.de>
Date: Sat, 06 Apr 2013 01:03:56 +0200
Subject: [Avr-ada-devel] Essential contribution to the build machinery
To: AVR-Ada <avr-ada-devel@lists.sourceforge.net>

I want to let you know that "Jedi" (avr-ada-devel@jedi.neoix.net) contributed a major clean-up of the make machinery in AVR-Ada.

I have just committed most of his changes. The commit log is:

clean up of the build machinery contributed by Jedi:

- clear structure of the Makefiles
- can build everything in avr tree without having to install the rts (works again)
- rts doesn't have to rebuild everything every time
- threads are now building
- "clean" only build tree
- "uninstall" implemented
- install explicitly installs directories and files that are necessary

Thank you very much, Jedi. This clean up was long overdue. It considerably lightens the work of the AVR-Ada developers.

From: Rolf Ebert <rolf.ebert.gcc@gmx.de>
Date: Sat, 18 May 2013 22:18:03 +0200
Subject: V1.2.2
To: AVR-Ada <avr-ada-devel@lists.sourceforge.net>

Changes in the V1.2.2 Release

- The **Run Time System** now has optimized assembler code to generate string images of integer variables. The code is based on new code of the upcoming avr libc 1.8.1. You can directly access the function as System.Int_Img.U32_Img.

- The **compile environment** (Makefiles, gpr-files) received a long due overhaul, contributed by Jedi.
- The **part specifications** for the primary devices (attiny2313 atmega8 atmega168 atmega169 atmega32 atmega328p atmega644p atmega2560) were regenerated from the latest available AVR Studio 4 release XML files.
- **AVR.Serial** is a drop in replacement for the existing AVR.UART. It actually is a renaming of a generic instantiation of the new AVR.Generic_Text_IO.
- **AVR.Generic_Text_IO** factors out the reusable parts of the input and output routines. You still have to provide routines for sending and receiving single bytes as generic parameters.
- **AVR.Strings.Text** is an adoption of Dmitry Kazakov's [Strings_Edit](http://www.dmitry-kazakov.de/ada/strings_edit.htm) packages for the small AVR processors.
- A new example shows the use of the relatively cheap humidity sensors DHT.
- A command interpreter makes use of the new support packages AVR.Serial and AVR.Strings.Text.

From: Roland Gaudig <roland.gaudig@gaudig.com>
Date: Mon, 20 May 2013 14:18:06 +0000
Subject: Experimental build AVR-Ada with gcc-4.8.0
To: avr-ada-devel@lists.sourceforge.net

This weekend I tried to compile AVR-Ada in combination with gcc-4.8.0 binutils-2.23.2 and avr-libc-1.8. The build went quite smoothly. There is only one conflict between gcc-4.8 and one patch and there is a problem with gnatmake.

As actual Debian or Ubuntu distributions are coming with gnat-gcc-4.6 I first created my own gnat-gcc-2.8. For building the avr-ada library also gprbuild has to be compiled for gcc-4.8.

I had to apply the following patch on gprbuild as gcc-4.8 complained about a style error.

gprbuild.patch :

```
################################
--- src/gpr_version.ads.orig    2013-05-19
    19:33:24.182847154 +0000
+++src/gpr_version.ads 2013-05-19
    19:33:46.418847247 +0000
  <at> <at> -24,7 +24,7  <at> <at>

package GPR_Version is
-   Gpr_Version: constant String := "2012";
+  Gpr_Version: constant String := "2012";
    --  Static string identifying this version
    function Gpr_Version_String return String;
################################
```

After this preparations I built the avr-gcc toolchain. I used the stock binutils without applying any patches.

On gcc-4.8 I applied the following patches, with line numbers adapted:

23-gcc-4.7-ada-Makefile.patch

24-gcc-4.7-ada-gnattools.patch

71-gcc-4.7-ada-freestanding.patch

73-gcc-4.7-ada-gnat1_print_path.patch

with 72-gcc-4.7-ada-timebase.patch I discovered a conflict inside gcc/ada/switch-c.adb with the new gcc version. Until gcc-4.7 avr-ada defined a -gnatet switch, which is now occupied by another function in gcc-4.8, so I had to rename it. I chose -gnateT to solve this conflict.

72-gcc-4.8.0-ada-timebase.patch :

[patch excerpt can be found in the original post in the list archives at https://lists.sourceforge.net/lists/listinfo/avr-ada-devel —plm]

I omitted these patches, as they are include in stock gcc-4.8:

25-gcc-4.7-ada-pr55243-nostamp.patch

no_25-gcc-4.7-ada-gnattools_bug55243.patch

I left the avr-ada sources unchanged and created just a symbolic link from gcc-4.7-rts to gcc-4.8-rts. Building avr-gcc and installing went fine without problems.

But compiling small examples fails, gnatmake exits with an exception:

```
###################################

Exception name: STORAGE_ERROR
Message: stack overflow or erroneous
         memory access

Exception STORAGE_ERROR raised, while
         processing project file
avr-gnatmake: "hello.gpr" processing failed

###################################
```

The next days I have to further investigate what the problem is.

This problem seems also be known to the GCC project: http://gcc.gnu.org/ml/gcc-testresults/2013-02/msg00112.html

By the way, the source code distribution avr-ada-1.2.2.tar.bz2 lacks the configure script. Therefore I used the one from the git repository.

Paraffin and Paraffinalia

From: Brad Moore <brad.moore@shaw.ca>
Date: Sat, 06 Apr 2013 12:20:11 -0600
Subject: ANN: Paraffin 4.0 and Paraffinalia for Ada 2012 with Ravenscar support and task pools
Newsgroups: comp.lang.ada

I am pleased to announce a new major release of Paraffin for Ada 2012.

Paraffin is a set of Ada 2012 generics that may be used to add parallelism to iterative loops and recursive code.

Paraffin now includes generics for both Ravenscar and non-Ravenscar use. The Ravenscar version utilizes static task pools with dispatching domains suitable for real-time programming.

Paraffin also includes Paraffinalia, which is a suit 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.
6) generic to perform prefix sum calculations.

The Ada 2012 version of the code is a major change to the API, which is why this is a major release. The 2005 version still has the old interface, and currently is treated as an archive, though the 2012 version of the API may eventually get backported to 2005 at some point.

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

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

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

```
mkdir sandbox
cd sandbox
git clone git://git.code.sf.net/p/paraffin/
code paraffin-code
```

The current development version typically will correspond to the latest stable release, but may at times be unstable when new features are being worked on.

Major New Features include:

- The new API is more unified between parallel loops and parallel recursion. A parallelism manager is declared at the site of the Parallelism OPportunity (POP).
- Ada 2012 support and use of Ada 2012 language features.
- Ravenscar generics added for parallel real-time programming. The Ravenscar generics require declaration of static worker task pools, since all tasks have to declared statically. The Ravenscar task pools can interoperate with Ada 2012 dispatching domains.
- Exception handling added to worker tasks which store the exception and reraise it in the main task before

returning from the parallelism opportunity

- Task pool versions also added for non-ravenscar. It was thought that task pools would introduce a performance benefit, but on the tested platforms this appears to not be the case, as creating tasks on the fly is approximately the same performance as reusing tasks from a task pool. Further, for non-ravenscar, using a bounded task pool introduces other undesirable affects centered around the case of handling when more tasks are requested than exist in the pool. Therefore, it is recommended for non-ravenscar use that the non-pooled versions of the generics be used. The pooled versions are provided for experimental purposes.
- No longer need to with target specific packages to use barriers, or affinity, since these are included in Ada 2012. This simplifies use in GNAT project files, since only the top level paraffin folder needs to be mentioned, if that folder is tagged to recursively pull in all the sub-folders in the project file.
- Many other changes, including improved folder structure.

GtkAda OpenGL demonstration applications

From: Francois Fabien <francois_fabien@hotmail.com>
Date: Fri, 12 Apr 2013 15:23:11 +0200
Subject: [gtkada] [ANN] OpenGL demos with GtkAda
To: gtkada <gtkada@lists.adacore.com>

3 demos of OpenGL with GtkAda are available for download:

- 2 lessons adapted from the NeHe tutorial
- a stand-alone penguin application, adapted from testgtk with some bug fix.

Source code is available at:

https://sourceforge.net/projects/lorenz/

For folks with Windows that do not have GtkAda installed, a bundle with all DLLs is also provided.

Qt5Ada

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Tue, 16 Apr 2013 04:48:01 -0700
Subject: Announce : Qt5Ada version 5.0.2 release 16/04/2013 free edition
Newsgroups: comp.lang.ada

Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.0.2 final) Qt5ada version 5.0.2 open source and qt5c.dll(libqt5c.so) built with Microsoft Visual Studio 2010 in Windows and gcc x86 in Linux Package tested with gnat gpl 2012 ada compiler in Windows 32bit and 64bit and Linux x86 Fedora 17 It supports GUI, SQL, Multimedia, Web, Network and many others thinks. Qt5Ada for Windows and Linux (Unix) is available from http://users1.jabry.com/adastudio/index.html

My configuration script to build Qt5 is: configure -opensource -release -nomake tests -opengl desktop -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -prefix "e:/Qt/5.0"

Plugins for database connections and DirectShow(needed for Camera example) are included.

I added many new packages, demos and small clip (10 min) in this release

Units of measurement

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 27 Apr 2013 15:23:03 +0200
Subject: ANN: Units of measurement v3.3
Newsgroups: comp.lang.ada

The library provides an implementation of dimensioned values for Ada. Unit checks are made at run-time, if not optimized out by the compiler. SI and irregular measurement units are supported. Shifted units like degrees Celsius are supported too. Conversions from and back to strings are provided for all various irregular units. An extensive set of GTK widgets for dealing with dimensioned values is included, though use of GTK is not mandatory for the rest of the library.

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

This release mixes minor bug in the package Measures_Irregular (the constant hp (Horsepower))

String clustering

From: Fumin <awawfumin@gmail.com>
Date: Mon Apr 29 22:28:05 CEST 2013
Subject: String clustering
URL: https://github.com/fumin/string-clustering

Ada

- To build the main program: make. This will create the executable "obj/main"
- Options of the program and how to run it:
  - First option: path to the data file (String).
  - Second option: number of classes (Integer).
  - Third option: whether to recompute the distance matrix (Boolean).
  - For example: obj/main ../sampled_keys_1000_over_1.txt 7 n
- To run tests: make test

[…]

Augusta

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Sun May 12 2013
Subject: Augusta
URL: https://github.com/pchapin/augusta

Augusta is an open source Ada 2012 compiler written in Scala that targets LLVM. This document is a quick description of how to build Augusta. For more information see the documentation in the 'doc' folder. The GitHub wiki for the project also includes some additional information for Augusta contributors.

Augusta is not even remotely usable at this time. However I do intend to keep the system in a buildable state so it should always be possible to create the Augusta jar file (and any associated files) as well as execute whatever tests have been written to date.

Augusta development is done on Linux (64 bit) and testing is done on both Windows 7 (64 bit) and Linux (64 bit). I expect it would be possible to do Augusta development on any system that supports the prerequisites. Mostly that means any system that supports Java and LLVM.

Adastudio

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Sun, 19 May 2013 01:16:22 -0700
Subject: ADASTUDIO 2013
Newsgroups: comp.lang.ada

Adastudio 2013 consists of two ISO files and contains:

- Qt5Ada
- Vtk5Ada
- vad
- Prebuilt QT5 (win32,x86)
- Prebuilt VTK 5.10.1 with QT5 (win32,x86)

Downloads:

- https://rapidshare.com/files/3231734174/Adastudio2013_1.iso
- https://rapidshare.com/files/3110901306/adastudio2013_2.iso

Alarm Clock using Arduino, DS1307, buzzer, and LCD

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Sun 19 May 2013
Subject: Alarm Clock using Arduino, DS1307, buzzer, and LCD
URL: http://arduino.ada-language.com/alarm-clock-using-arduino-ds1307-buzzer-and-lcd.html

Writing this article took somewhat longer than usual, since the scope expanded on the way. Originally, I planned to only show how to use a buzzer, but then I

decided to add a DS1307 real-time clock chip to it. And of course, you need a LCD display for showing the time. But unluckily, my previous LCD shield was using conflicting pins with other stuff, so I had to redo it also.

In the end, this was the result:

http://farm9.staticflickr.com/8539/8680908563_c073efd99f.jpg

There are three shields in the picture: buzzer shield, DS1307 shield, and LCD shield. And below them is Olimexino-328, an Arduino clone. You can also use normal Arduino, but I had one Olimexino-328 at hand, so I used it.

[Tero provides us with instructions and source text for building an Arduino based alarm clock. —sparre]

ASN.1

From: Manuel Gomez <mgrojo@gmail.com>
Date: Wed, 22 May 2013 22:31:19 +0200
Subject: Re: ASN.1 to Ada?
Newsgroups: comp.lang.ada

> Are there any Open Source ASN.1 to Ada compilers out there?

Have you looked at this? It seems to meet your requirements, but I haven't used it myself:

http://taste.tuxfamily.org/wiki/index.php?title=ASN.1_generators

Ada-related Products

Atego ApexAda Developer for Ada 2005

From: Atego Press Releases
Date: Tue Mar 12 2013
Subject: Atego Releases New Version of Atego ApexAda Developer for Ada 2005
URL: http://www.atego.com/pressreleases/pressitem/atego-releases-new-version-of-atego-apexada-developer-for-ada-200

Atego ApexAda Developer 5.0 offers complete support for Ada 2005 passing all available Ada conformity assessment test suite tests.

Atego™, the leading independent supplier of industrial-grade, collaborative development tools for engineering complex, mission- and safety-critical architectures, systems, software and hardware, launches a new version of Atego ApexAda™ Developer Enterprise Edition with Ada 2005 support.

Atego ApexAda Developer 5.0 is a major new release with complete support for the language features in Ada 2005. It passes all available Ada Conformity Assessment Test Suite (ACATS) tests applicable for native code compilation and execution. The Atego ApexAda Developer product provides a complete solution for editing, compiling, debugging, testing, managing, and executing even the most complex and challenging Ada applications on UNIX and Linux-based operating systems.

In addition to traditional compilation features, Atego ApexAda Developer incorporates a sophisticated architectural and build control mechanism, configuration management and version control (CMVC) capabilities to help development teams keep control of their updates and manage changes to their application code. Atego TestMate™ is also included to provide automation of the testing process, including regression testing and coverage analysis. In addition to being an integration and system testing tool to help automate the software testing process, Atego TestMate also allows developers to easily compose test cases into test suites that can be executed and evaluated automatically.

For projects requiring support of the X-Windows System's network display services and the OSF/Motif™ graphical user interface (GUI), Atego provides AXI™ Bindings for Atego ApexAda Developer 5.0 as an add-on product. AXI gives Ada programmers access to more than 2,000 functions and types in the X library, extensions and utilities, the Xt toolkit and the Motif widget set and resource manager.

"This release advances Atego's long term commitment to Ada its Ada customers which continue to utilize Ada as an invaluable asset in their development arsenal," stated Hedley Apperly, Atego's Vice-President of Product & Marketing. "Atego will continue to invest in further enhancements to the Atego ApexAda Developer line with embedded and real-time products for Ada 2005 now and in the future."

Shipping and Availability

Atego ApexAda Developer 5.0 is fully released and immediately available for native code development and execution on Solaris/SPARC, Solaris/Intel, Linux/Intel platforms. Atego ApexAda Developer 5.0 supports RedHat, SUSE, and other popular Linux distributions. Special product license pricing is available for existing Atego ApexAda Developer users

AdaCore releases GNAT Pro 7.1

From: AdaCore Press Center
Date: Tue Apr 23 2013
Subject: AdaCore Releases GNAT Pro 7.1
URL: http://www.adacore.com/press/adacore-releases-gnat-pro-71/

Latest Ada development environment brings full Ada 2012 support and other enhancements.

SAN JOSE, Calif., NEW YORK and PARIS, April 23, 2013 – Design West Conference – AdaCore today announced the availability of GNAT Pro 7.1, a major new version of the company's flagship development environment. This release incorporates a number of enhancements, many based on user suggestions, including several Ada language related features, run-time improvements, and new and enhanced tools. As with all AdaCore products, GNAT Pro is Freely-Licensed Open Source Software (FLOSS).

The language related features in GNAT Pro 7.1 include updated and improved support for the Ada 2012 language revision, a new facility for automatic data endianness conversion, and support for dimensionality checking. Extended overflow check support and lock-free protected objects have been incorporated as run-time improvements, and the compiler technology has been migrated to the GCC 4.7 back-end.

New and enhanced tools that aid in all stages of the development of complex, multi-language software are also a major part of the GNAT Pro 7.1 release. These include the powerful and easy to use GPRBuild 2.0 tool that automates the construction of multi-language software, the GDB 7.5 debugger and new rules in the GNATcheck coding standard verification tool. GNAT Pro 7.1 also includes a major new release of AdaCore's high-level graphical toolkit GtkAda, which provides a secure, user-friendly and extensible toolkit based on Gtk+. GtkAda 3 brings new widgets, a CSS based theming framework, and an improved application programming interface (API) that is more intuitive and that incorporates a more homogenous naming scheme.

"As usual, the new GNAT Pro release brings a combination of advantages: extreme stability for existing users, support for an increasingly wider range of platforms and targets, significant performance enhancements, and a wealth of new capabilities and features," said Cyrille Comar, Managing Director of AdaCore. "The most striking new feature, support for automatic data endianness conversion, is already a customer favorite; who said that silver bullets were mythical?"

About GNAT Pro

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

backed by rapid and expert support service.

Pricing and Availability

GNAT Pro 7.1 is available immediately on most supported platforms. Please contact AdaCore (sales@adacore.com) for further details on pricing and supported configurations.

AdaCore releases major new version of CodePeer static analysis tool

From: AdaCore Press Center
Date: Tue Apr 23 2013
Subject: AdaCore Releases Major New Version of CodePeer Static Analysis Tool
URL: http://www.adacore.com/press/adacore-releases-major-new-version-of-codepeer-static-analysis-tool/

SAN JOSE, Calif., NEW YORK and PARIS, April 23, 2013 – Design West Conference – AdaCore today announced the release of CodePeer 2.2, the advanced static analysis tool that helps developers detect potential run-time and logic errors in Ada programs. CodePeer is able to find non-trivial problems by systematically analyzing every possible input and path through the program, and can be employed very early in the development cycle to identify defects when they are the least costly to repair.

CodePeer is fully integrated into the GNAT Pro development environment and comes with a number of complementary static analysis tools common to the technology – a coding standard verification tool (GNATcheck), a source code metric generator (GNATmetric), a semantic analyzer and a document generator.

CodePeer 2.2 introduces many improvements, driven by customer feedback, including:

- Integration into GNATbench, the GNAT Pro Ada plug-in for Eclipse and Wind River Systems Workbench
- Full support for GNAT project files
- Message review from HTML reports
- New "–level" switch to easily tune CodePeer messages and analysis time for any kind of code base
- More accurate analysis of math functions and floating point computations
- Export of messages to spreadsheets

"CodePeer 2.2 brings users an extra level of flexibility through interfaces to Eclipse, spreadsheets, and full HTML capabilities," said Arnaud Charlet, CodePeer Product Manager at AdaCore. "This new release now provides solutions for the full range of Ada projects, including those with requirements for the highest levels of integrity and certification, as well as systems with large and complex code bases."

Demonstration

A pre-recorded demo presented by Quentin Ochem introducing the latest CodePeer 2.2 features is currently available online. Please visit: http://www.adacore.com/knowledge/demos/codepeer-2-2

About CodePeer

Serving as an efficient and accurate code reviewer, CodePeer identifies constructs that are likely to lead to run-time errors such as buffer overflows, and it flags legal but suspect code, typical of logic errors. Going well beyond the capabilities of typical static analysis tools, CodePeer also produces a detailed analysis of each subprogram, including pre- and post-conditions. Such an analysis makes it easier to find potential bugs and vulnerabilities early: if the implicit specification deduced by CodePeer does not match the component's requirements, a reviewer is alerted immediately to a likely logic error. During system development, CodePeer can help prevent errors from being introduced, and it can also be used as part of a systematic code review process to dramatically increase the efficiency of human review. Furthermore, CodePeer can be used retrospectively on existing code, to detect and remove latent bugs.

Pricing and Availability

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

Ada and GNU/Linux

AVR-Ada RPMs for Fedora

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Fri, 8 Mar 2013 22:20:41 +0200
Subject: Unofficial AVR-Ada RPMs for Fedora 17 and 18 available
Newsgroups: comp.lang.ada

To make AVR-Ada installation easier, I made AVR-Ada RPM packages for Fedora 17 and 18. They are available for i386 and x86_64 architectures.

To install them, run following commands:

```
sudo wget -O /etc/yum.repos.d/
fedora-adalanguage.repo http://fedora.ada-
language.com/
fedora-adalanguage.repo
sudo yum install --nogpgcheck avr-adalib
```

Package "avr-ada-lib" will get all dependencies to be installed automatically also.

If you have "avr-gcc" package installed, please remove it first, since my "avr-gnat" package conflicts with it (and provides equal functionality).

The packages are based on AVR-Ada repository revision 8586204ed, which is AVR-Ada 1.2 + some patches. The source tar ball of that revision can be found inside source rpms[1].

And if you do not trust my binaries and want to build the rpm packages by yourself, you can get the .spec files from my Mercurial repositories:

- https://bitbucket.org/tkoskine/fedora-avr-ada-rts
- https://bitbucket.org/tkoskine/fedora-avr-ada-lib
- https://bitbucket.org/tkoskine/fedora-avr-gcc

To test the packages, you can use example code from my arduino-hello-uart repo:

```
hg clone https://bitbucket.org/tkoskine/
arduino-hello-uart
cd arduino-hello-uart
make
```

At some point, I might try to get the packages included to Fedora distribution, but for now they are offered separately from my servers.

Debian 7.0

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon, 20 May 2013
Subject: Ada applications, compilers and libraries in Debian 7.0 Wheezy

Earlier this month Debian 7.0 Wheezy was released. It provides the GNAT 4.6 Ada compiler for the platforms:

- amd64
- armel
- armhf
- hurd-i386
- i386
- ia64
- kfreebsd-amd64
- kfreebsd-i386
- mips
- mipsel
- powerpc
- ppc64
- s390
- sparc

The following add-ons for Ada programmers are also provided:

- The Ada 2012 Reference Manual
- AdaBrowse 4.0.3
- AdaCGI 1.6
- AdaControl 1.12r3
- APQ 3.2
- AdaSockets 1.8.10

- Ahven 2.1
- Alog 0.4.1
- anet 0.1
- ASIS 2010
- AUnit 1.03
- AWS 2.10.2
- D-bus for Ada 0.2
- Florist 2011
- GDB 7.4.1
- GMPAda 0.0.20120331
- GNADE 1.6.2
- GPRBuild 2011
- GPS 5.0
- GtkAda 2.24.1
- Log4Ada 1.2
- Ncurses for Ada 5.9
- OpenToken 4.0b
- PC/SC Ada 0.7.1
- PolyORB 2.8 prerelease
- PLPlot 5.9.5
- SPARK 2011
- Templates Parser 11.6
- XML/Ada 4.1
- XML-EZ-out 1.06.1

Some included (non-software-development) applications written in Ada:

- music123 -- command-line shell for sound-file players.
- topal -- links Pine and GnuPG together.

Ada and Mac OS X

Calling Objective-C classes

From: Pascal <sur.pignard-@wannado.fr>
Date: Mon, 25 Mar 2013 22:14:44 +0100
Subject: XCode/Objective-C with GNAT/Ada.
Newsgroups: gmane.comp.lang.ada.macosx

Hello, here is my first progress status of calling Objective-C classes with Ada.

Since Apple had moved to Objective-C version 2.0, the bindings Cocoa-Gnat from Bill Greene have been no more working on MacOS 10.8. (http://code.google.com/p/cocoa-gnat)

Thus, I've changed of point of view in order to simplify the problem: I've started with XCode. I've written an Objective-C class with instance variables, instance methods and one class method. Then I've built an extern Ada library with GNAT containing an Ada sub-program taking an Objective-C instance of my class as parameter. Thanks to Apple runtime in C, the Ada part calls instance and class method via runtime functions.

See class interface:

```
 <at>interface Classe01 : NSObject
{
   int n;
   float factorielle;
}
 <at>property int n;
 <at>property float factorielle;
+ (void) quiSuisJe;
- (void) calculFactorielle;
 <at>end
```

See main program:

```
// insert code here...
/* First, elaborate the library before using it */
bibliinit (); -- GNAT init
Classe01 *calcul = [[Classe01 alloc] init];
[Classe01 quiSuisJe];
Class cl01 = objc_getClass("Classe01");
NSLog( <at>"Il s'agit de la classe %s",
          class_getName(cl01));
SEL swai = sel_registerName("quiSuisJe");
NSLog( <at>"Il s'agit du sélecteur %s",
          sel_getName(swai));
Method mwai = class_getClassMethod
          (cl01, swai);
IMP fwai = method_getImplementation
          (mwai);
fwai(cl01, swai);
[calcul setN:20];
[calcul calculFactorielle];
NSLog( <at>"Factorielle %d : %f", [calcul n],
          [calcul factorielle]);
/* Main program, using the library
   exported entities */
affiche (); -- Ada call
[calcul setN:10];
calcfact (calcul); -- Ada call
NSLog( <at>"Factorielle %d : %f", [calcul n],
          [calcul factorielle]);
calcfact2 (); -- Ada call
/* Library finalization at the end
   of the program */
biblifinal (); -- GNAT final
```

See Ada sub-programs:

```
procedure Affiche is
begin
   Put_Line ("Bibli1.Affiche");
end Affiche;
type SEL is new System.Address;
type Class is new System.Address;
type Method is new System.Address;
type IMP is access procedure (aClass :
          Class; aSelector : SEL);
pragma Convention (C, IMP);
function objc_getClass (name : chars_ptr)
          return Class;
pragma Import (C, objc_getClass,
          "objc_getClass");
function class_getName (cls : Class)
          return chars_ptr;
pragma Import (C, class_getName,
          "class_getName");
function class_getClassMethod
    (aClass    : Class;
     aSelector : SEL)
     return Method;
pragma Import (C, class_getClassMethod,
          "class_getClassMethod");
function method_getImplementation
          (aMethod : Method) return IMP;
pragma Import (C,
          method_getImplementation,
          "method_getImplementation");
function object_getClassName (obj : id)
          return chars_ptr;
pragma Import (C, object_getClassName,
          "object_getClassName");
function sel_registerName (str : chars_ptr)
          return SEL;
pragma Import (C, sel_registerName,
          "sel_registerName");
function sel_getName (aSelector : SEL)
          return chars_ptr;
pragma Import (C, sel_getName,
          "sel_getName");
function objc_msgSend (theReceiver : id;
          theSelector : SEL) return id;
pragma Import (C, objc_msgSend,
          "objc_msgSend");

procedure CalcFact (This : id) is
    Dum_ID : id;
  begin
    Put_Line (System.Address_Image
          (System.Address (This)));
    Put_Line (Value (object_getClassName
          (This)));
    Put_Line(Value
        (sel_getName
          (sel_registerName (New_String
                ("calculFactorielle")))));
    Dum_ID := objc_msgSend(This,
        sel_registerName (New_String
                ("calculFactorielle")));
  end CalcFact;
procedure CalcFact2 is
  cl01 : constant Class  := objc_getClass
          (New_String ("Classe01"));
  swai : constant SEL    :=
          sel_registerName (New_String
          ("quiSuisJe"));
  mwai : constant Method :=
          class_getClassMethod (cl01, swai);
  -- attention conflit avec fwai de main.m
  fwai2 : constant IMP :=
          method_getImplementation (mwai);
begin
  Put_Line (Value (class_getName (cl01)));
  Put_Line (Value (sel_getName (swai)));
  fwai2 (cl01, swai);
end CalcFact2;
```

CalcFact calls instance method and CalcFact2 calls class method.

Result:

```
2013-03-25 21:47:21.723
          Essai01[10062:303] Je suis la
          classe Classe01.
2013-03-25 21:47:21.726
          Essai01[10062:303] Il s'agit de la
          classe Classe01
2013-03-25 21:47:21.726
          Essai01[10062:303] Il s'agit du
          sélecteur  quiSuisJe
2013-03-25 21:47:21.727
          Essai01[10062:303] Je suis la
          classe Classe01.
2013-03-25 21:47:21.727
          Essai01[10062:303] Factorielle 20 :
          2432902023163674624.00000
```

```
Bibli1.Affiche
00000001001082C0
Classe01
calculFactorielle
2013-03-25 21:47:21.728
        Essai01[10062:303] Factorielle 10 :
        3628800.000000
Classe01
quiSuisJe
2013-03-25 21:47:21.728
        Essai01[10062:303] Je suis la
        classe Classe01.
```

The result with Ada calls is the same as with Objective-C calls ;-)

I find this really promising. Next steps are to allocate a new instance from Ada, call Cocoa classes**...**

GCC 4.8.0 for Mac OS X

From: Simon Wright <simon@pushface.org>
Date: Mon, 15 Apr 2013 08:38:38 +0100
Subject: GCC 4.8.0 for Mac OS X
Newsgroups: comp.lang.ada

You can find this at

https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/4.8.0/

The README says:

This is GCC 4.8.0 built for Mac OS X Mountain Lion.

Includes ASIS, AUnit, GPRbuild, XMLAda from GNAT GPL 2012.

Compilers included: Ada C C++ Fortran.

Target: x86_64-apple-darwin12

Configured with:

```
../gcc-4.8.0/configure \
  --prefix=/opt/gcc-4.8.0 \
  --disable-multilib \
  --enable-languages=c,ada,c++,fortran \
  --target=x86_64-apple-darwin12 \
  --build=x86_64-apple-darwin12
```

Thread model: posix

gcc version 4.8.0 (GCC)

Install by

```
  $ cd /
  $ sudo tar jxvf ~/Downloads/
gcc-4.8.0-x86_64-apple-darwin12.tar.bz2
```

and put /opt/gcc-4.8.0/bin first on your PATH.

MD5 (gcc-4.8.0-x86_64-apple-darwin12.tar.bz2) = db2c8b196475faa648b8f40b66465692

From: Simon Wright <simon@pushface.org>
Date: Tue, 30 Apr 2013
Subject: Building GCC 4.8.0
URL: http://forward-in-code.blogspot.co.uk/2013/04/building-gcc-480.html

These notes describe building GCC 4.8.0 for Mac OS X, with Ada, C, C++, Fortran, Objective C, Objective C++, and various GNAT tools.

Build environment

I'm building on a 13" Macbook Pro with a 2.5 GHz Intel Core 2 i5 processor and 4 GB of RAM, running Mac OS X Mountain Lion 10.8.3 (Darwin 12.3.0) with Xcode 4.6.2.

[…]

[Simon gives detailed instructions on building GCC 4.8.0 (including GNAT) on Mac OS X. —sparre]

Ada and Microsoft

How to get screen size

From: Tom Moran <tmoran@acm.org>
Date: Sat, 6 Apr 2013 17:56:12 +0000
Subject: Re: How to get the screen size?
Newsgroups: comp.lang.ada

> To make my ada application program suitable for more monitor sizes, I need the Screen Size. Where the size is the number of pixels. Is there a function or procedure to get the screen width and height?

CLAW (Class Library for Ada on Windows) has Get_System_Metrics, which basically calls Windows' GetSystemMetrics with the right parameter.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 7 Apr 2013 12:00:24 +0200
Subject: Re: How to get the screen size?
Newsgroups: comp.lang.ada> The OS is Windows 7.

Anyway, if your application is limited to Windows already, there is no reason not to use Win32Ada.

You call GetDesktopWindow from the package Win32.Winuser, which gives you a handle to the desktop window. Then you call GetWindowRect on that window. The rectangle returned is one of the desktop. For further information see MSDN:

http://msdn.microsoft.com/en-us/library/windows/desktop/ms633504%28v=vs.85%29.aspx

Note than Windows supports multiple monitors. You could wish to enumerate them. See GetMonitorInfo in MSDN.

References to Publications

Optimization with Valgrind Massif and Cachegrind

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: March 2 2013
Subject: Optimization with Valgrind Massif and Cachegrind
URL: http://blog.vacs.fr/index.php?post/2013/03/02/Optimization-with-Valgrind-Massif-and-Cachegrind

Memory optimization reveals sometimes some nice surprise. I was interested to analyze the memory used by the Ada Server Faces framework. For this I've profiled the unit tests program. This includes 130 tests that cover almost all the features of the framework.

Memory analysis with Valgrind Massif

Massif is a Valgrind tool that is used for heap analysis. It does not require the application to be re-compiled and can be used easily. The application is executed by using Valgrind and its tool Massif. […]

The valgrind tool creates a file massif.out.NNN which contains the analysis. The massif-visualizer is a graphical tool that reads the file and allows you to analyze the results. […]

Introduction to AWS (part 2)

From: Thomas Løcke <tl@ada-dk.org>
Date: March 10, 2013
Subject: Using the Ada Web Server (AWS), part 2
URL: http://blogs.fsfe.org/thomaslocke/2013/03/10/using-the-ada-web-server-aws-part-2/

In the Using the Ada Web Server (AWS), part 1 article I showed you how to setup a simple Hello world! server powered by the awesomeness that is the Ada Web Server (AWS) project. In this second part I will show you how to utilize the Templates_Parser module to build your HTML and I'll also give a very short example on how to serve a flat file to visitors.

[…]

"Programming in Ada 2012" expected in 2014

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 27 Mar 2013 13:58:45 -0500
Subject: Re: Ada 2005 vs Ada 2012
Newsgroups: comp.lang.ada

Here's the definitive answer from John Barnes that he asked me to post:

"John is planning to write Programming in Ada 2012 but he has not yet finalized the contract with the publishers; he would not expect it to be published before early 2014."

He also said that he's planning to finish the Rationale for Ada 2012 before starting on a revision of the book. So it looks like quite a wait for the book.

Dr.Dobb's: Ada With Contracts

From: Robert Dewar
Date: Tue Apr 9 2013
Subject: Ada 2012: Ada With Contracts
URL: http://www.drdobbs.com/architecture-and-design/ada-2012-ada-with-contracts/240150569

The most important new feature in Ada 2012 is support for contract-based programming, which adds more validation of mission-critical code to a language already famous for its focus on reliability.

The most recent version of the Ada standard, known as Ada 2012, brings contract-based programming to a mainstream language. Preconditions, postconditions, type invariants, and subtype predicates allow software developers to specify their programs' intent more clearly, in effect embedding requirements into the source code. This feature makes it easier to review and understand the program, and it facilitates verification by the compiler, static analysis tools, and/or runtime checks.

[…]

Comparing YAMI4 and ZeroMQ

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 23 Apr 2013 14:04:04 -0700
Subject: YAMI4 vs. ZeroMQ
Newsgroups: comp.distributed

The following article compares YAMI4 to ZeroMQ:

http://www.inspirel.com/articles/YAMI4_vs_ZeroMQ.html

Both libraries offer messaging solutions for distributed system developers, but they differ quite heavily on several policies. The above article presents the YAMI4 point of view, but with the intention of being honest and accurate.

Static predicates

From: Thomas Løcke <tl@ada-dk.org>
Date: April 25, 2013
Subject: Su(per)btypes in Ada 2012 – Part 1
URL: http://ada-dk.org/2013/04/superbtypes-in-ada-2012-part-1/

If you want to learn a bit about types and the Static_Predicate aspect in Ada 2012, then this AdaCore gem has the good stuff:

> Ada 2012 is full of features for specifying a rich set of type properties. In this series of three Gems, we describe three aspects that can be used to state invariant properties of types and subtypes. This first Gem is concerned with the Static_Predicate aspect.

With the Static_Predicate aspect you can do cool stuff like this:

```
type Day is (Monday, Tuesday, Wednesday,
      Thursday, Friday, Saturday, Sunday);
type T_Day is new Day with
         Static_Predicate => T_Day in
         Tuesday | Thursday;
```

Nice eh'? The compiler will now protect you against assigning Friday to a T_Day. That is pretty neat. Stay tuned for part 2 and 3, where more of the Ada 2012 goodness will be explained.

Article in German magazine Heise

From: Peter Dencker <peter.dencker@etas.com>
Date: Thu, 23 May 2013 12:49:23 +0000
Subject: Ada - Artikel auf heise
To: Liste der Fachgruppe Ada <ada@gi-fb-sicherheit.de>

Ein Ada Artikel auf heise-online zu Ihrer Information:

http://www.heise.de/developer/artikel/Nebenlaeufige-Programmierung-in-Ada-1862433.html

[Good publicity for Ada 2012. —sparre]

Ada Inside

IKEv2 Trusted Key Manager

From: Adrian-Ken Rueegsegger <ken@codelabs.ch>
Date: Mon, 08 Apr 2013 15:40:59 +0200
Subject: ANN: IKEv2 Trusted Key Manager
Newsgroups: comp.lang.ada

We are proud to announce that the popular strongSwan open source IPsec VPN project now makes use of Ada.

The latest strongSwan release 5.0.3 [1] contains support for the Trusted Key Manager (TKM), which is a separate process providing security critical operations of the IKEv2 protocol. The TKM has been implemented from scratch in Ada.

Further information about the IKEv2 disaggregation concept, the design and implementation of the TKM is available on the project website [2].

[1] http://www.strongswan.org/blog/2013/04/06/strongswan-5.0.3-released.html

[2] http://www.codelabs.ch/tkm/

First person shooter

From: Thomas Løcke <tl@ada-dk.org>
Date: April 16, 2013
Subject: AdaDoom3 – FPS Action in Ada
URL: http://ada-dk.org/2013/04/adadoom3-fsp-action-in-ada/

Those of us who are gamers at heart, know that Id Software usually release their "old" games and tech as open source, and naturally that also happened for the Doom 3 game. This in itself is good news, but what is even better is the fact that efforts are being put into an Ada port. The man behind the project is only known to me as J JS [1], and he is hosting the project at the AdaDoom3 [2] GitHub repository.

A few posts have been made about the project at the Ada Programming Community [3]:

- Initial news item about AdaDoom3 [4], by Thomas Løcke [5]
- State of AdaDoom3 [6], by Adam Wolk [7]

Feel free to pop in and join the Doom fun!

[1] https://plus.google.com/u/0/104228556547212920341/posts

[2] https://github.com/AdaDoom3/AdaDoom3

[3] https://plus.google.com/u/0/communities/102688015980369378804

[4] https://plus.google.com/u/0/112815721307813813920/posts/EszZMWbYu32

[5] https://plus.google.com/u/0/112815721307813813920

[6] https://plus.google.com/u/0/102090077582777383295/posts/StaiiW72Czw

[7] https://plus.google.com/u/0/102090077582777383295

AZip - A portable Zip archive manager

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sat 11 May 2013
Subject: AZip 1.20 - tree view
URL: http://gautiersblog.blogspot.dk/2013/05/azip-120-tree-view.html

http://azip.sf.net

[And a screenshot. — sparre]

Indirect information on Ada usage

From: David Dhénaux
Date: Wed May 15 2013
Subject: Paris - Ingéeur Logiciel Java ou Ada

Fondé par quatre ingéneurs, SmartSide est une jeune entreprise en plein essor, proposant un système d'information innovant dédié aux problématiques des Smart Grids.

Votre missions:

Vous participez à la réalisation des nouvelles fonctionnalités et intervenez sur toutes les activités inhérentes au développement logiciel.

Vous et SmartSide:

Vous aimez travailler en équipe et êtes motivé à l'idée de rejoindre une jeune

entreprise innovante afin de participer à son essor. Vous êtes attiré par les équipes dynamiques et cohésives dans lesquelles l'entraide et le partage de connaissances sont des valeurs primordiales. Vous aimez les défis techniques et vous engager pour les relever. Vous avez le sens de l'humour. Vous êtes intéressé, initié ou grand fanatique des méthodes Agiles. Votre aisance relationnelle à communiquer vos idées, votre esprit d'équipe et votre capacité d'analyse sont vos atouts majeurs.

Vous intégrerez une équipe à taille humaine qui travaille dans la bonne humeur et le respect d'autrui. Ensemble, nous partagerons des valeurs essentielles concernant notamment le bien-être de chacun. La communication avec les dirigeants sera simple et vous ne devrez pas attendre des mois pour qu'une décision soit prise. L'équipe sera à votre écoute et nous étudierons ensemble toutes vos propositions avec attention, qu'elles soient techniques ou relationnelles.

Vos Compétences : 2 à 5 ans d'expérience, BAC +5, Java et/ou Ada, méthodes Agiles.

Ada in Context

Automatic parallelism

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Fri, 08 Mar 2013 11:18:48 +0100
Subject: Re: Ada and OpenMP
Newsgroups: comp.lang.ada

> Fortunately, OpenMP is no longer needed to achieve automatic parallelism in either C or Ada at the low level. GCC's vectorizer produces code that runs in parallel for a number of loop patterns. These are documented, and they work in GNAT GPL or more recent FSF GNATs. Later 4.7s IIRC.

For example, adding -ftree-vectorize to the set of options (-O2 …) increases the speed of the program below by factors up to 3, depending to some extent on the value of MAX. (Option -O3 is even easier in this case, and yields improvements when MAX = 8.)

The assembly listing includes instructions like MOVDQA and PADDD used with SSE registers.

GNAT will report successful optimizations when -fopt-info-optimized is among the switches (or -ftree-vectorizer-verbose=2 for older GNATs).

```
package Fast is
   MAX : constant := 50;

   subtype Number is Integer;
   type Index is new Natural range 0 .. MAX;
   type Vect is array (Index) of Number;

   procedure Inc_Array (V : in out Vect);
end Fast;

package body Fast is
   procedure Inc_Array (V : in out Vect) is
   begin
      for K in Index loop
         V (K) := V (K) + 1;
      end loop;
   end Inc_Array;
end Fast;

with Ada.Real_Time;    use Ada.Real_Time;
with Ada.Text_IO;
with Fast;    use Fast;
procedure Test_Fast is
   Start, Finish : Time;
   Data : Vect;
   Result : Integer := 0;
   pragma Volatile (Result);
begin
   Start := Clock;
   for Run in 1 .. 500_000_000/MAX loop
      Inc_Array (Data);
      if Data (Index(MAX/2 + Run mod
            MAX/2)) rem 2 = 1 then
         Result := 1;
      end if;
   end loop;
   Finish := Clock;
   Ada.Text_IO.Put_Line
     (Duration'Image (
           To_Duration (Finish -Start)));
end Test_Fast;
```

Bulletin from the Anti-Pragma Society

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 7 Mar 2013 17:42:59 -0600
Subject: Re: Ada and OpenMP
Newsgroups: comp.lang.ada

> In Ada one might write, perhaps
>
> pragma OMP(Parallel_For)
>
> for I in 1 .. MAX loop
>
> A(I) := A(I) + 1
>
> end loop;
>
> Doing this with Ada tasks in such a way that it uses an optimal number of threads on each execution (based on core count) would be much more complicated, I should imagine. Please correct me if I'm wrong!

Well, this doesn't make much sense. If the pragma doesn't change the semantics of the loop, then it is not necessary at all (the compiler can and ought to do the optimization when it makes sense, possibly under the control of global flags). Programmers are lousy at determining where and how the best of use of machine resources can be made. (Pragma Inline is a similar thing that should never have existed and certainly shouldn't be necessary.)

If the pragma does change the semantics, then it violates "good taste in pragmas". It would be much better for the change to be indicated by syntax or by an aspect.

Pragmas, IMHO, are the worst way to do anything. Compiler writers tend to use them because they can do so without appearing to modify the language, but it's all an illusion: the program probably won't work right without the pragma, so you're still locked into that particular vendor. Might as well have done it right in the first place (and make a proposal to the ARG, backed with practice, so it can get done right in the next version of Ada).

Randy Brukardt,

President, Anti-Pragma Society. :-)

Case conversion of UTF-8 encoded Unicode strings

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 7 Mar 2013 17:53:25 -0600
Subject: Re: string and wide string usage
Newsgroups: comp.lang.ada

[...]
Right. The proper thing to do (for Ada 2012) is to use Ada.Characters.Wide_Handling (or Wide_Wide_Handling) to do the case conversion, after converting the UTF-8 into a Wide_String (or Wide_Wide_String).

If you're trying to do this in an older version of Ada, you'll have to find some library somewhere to do the job.

But I want to caution you that "converting to lower case" is not a great idea if you plan to support arbitrary Unicode strings. Such conversions are somewhat ambiguous, and tend to make strings appear similar that are different (and sometimes the reverse happens as well). Usually, the best plan is to store the strings unmodified and use Equal_Case_Insensitive to compare them (this uses the most accurate comparison defined by Unicode, and has the advantage of being guaranteed not to change in future character set standards, which is NOT true of conversion to lower case).

There is a nice example of this problem in the next chapter of the Ada 2012 Rationale (although you'll have to wait until May to see it, unless you get the Ada User Journal).

I realize you may have no choice given the design of your database might not be in your control, and it might not matter if you don't plan to have Greek and Turkish characters in your data (to mention two of the most common where convert to lower case and Equal_Case_Insensitive give different answers for Wide_Strings).

Parallelised execution of loops

From: Georg Bauhaus <bauhaus@futureapps.de>

Date: Fri, 08 Mar 2013 16:47:06 +0100
Subject: Re: Ada and OpenMP
Newsgroups: comp.lang.ada

> for I in 1 .. MAX loop
>
> A(I) := A(I) + 1
>
> end loop;

If A is an array of small objects, then with GNAT on Intel, you can turn on -ftree-vectorize (or -O3) and see what this gives.

Adding -ftree-vectorizer-verbose=2 (old) or -fopt-info-optimize instructs GCC to report successful vectorizations.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 8 Mar 2013 17:40:21 -0600
Subject: Re: Ada and OpenMP
Newsgroups: comp.lang.ada

(1) Use a compiler that does this automatically (apparently GNAT does this in some circumstances).

(2) Use a library like Paraffin; a bit less convenient but it will work on any Ada compiler for any target. Some of the Ada 2012 features may make such a library more convenient to write (I haven't been keeping up with Brad's work on this).

(3) Use a compiler with an appropriate extension for parallel loops. One possibility would be something like:

```
for I in 1 .. MAX loop in parallel
  A(I) := A(I) + 1
end loop;
```

This of course ties you to a particular implementation, or to wait for Ada 202x. Of course, so does a pragma, and it's much less likely to be standardized. So I suggest (1) or (2).

From: Brad Moore
<brad.moore@shaw.ca>
Date: Fri, 08 Mar 2013 14:37:04 -0700
Subject: Re: Ada and OpenMP
Newsgroups: comp.lang.ada

The following code shows the same problem executed sequentially, and then executed with Paraffin libraries.

```
with Ada.Real_Time;    use Ada.Real_Time;
with Ada.Command_Line;
with Ada.Text_IO; use Ada.Text_IO;
with Parallel.Iteration.Work_Stealing;

procedure Test_Loops is

   procedure Integer_Loops is new
     Parallel.Iteration.Work_Stealing
           (Iteration_Index_Type => Integer);

   Start : Time;

   Array_Size : Natural := 50;
   Iterations : Natural := 10_000_000;

begin

   --  Allow first command line parameter to
   -- override default iteration count
   if Ada.Command_Line.
            Argument_Count >= 1 then
      Iterations := Integer'Value
         (Ada.Command_Line.Argument (1));
   end if;

   --  Allow second command line parameter
   -- to override default array size
   if Ada.Command_Line.
            Argument_Count >= 2 then
      Array_Size := Integer'Value
         (Ada.Command_Line.Argument (2));
   end if;

   Data_Block : declare
      Data : array (1 .. Array_Size)
            of Natural := (others => 0);
   begin

      -- Sequential Version of the code, any
      -- parallelization must be auto
      -- generated by the compiler

      Start := Clock;

      for I in Data'Range loop
         for J in 1 .. Iterations loop
            Data (I) := Data (I) + 1;
         end loop;
      end loop;

      Put_Line ("Sequential Elapsed=" &
            Duration'Image (To_Duration
                  (Clock - Start)));

      Data := (others => 0);
      Start := Clock;

      --  Parallel Version of the code, explicitly
      -- parallelized using Paraffin

      declare

         procedure Iterate (First : Integer;
                            Last : Integer) is
         begin
            for I in First .. Last loop
               for J in 1 .. Iterations loop
                  Data (I) := Data (I) + 1;
               end loop;
            end loop;
         end Iterate;

      begin
         Integer_Loops (
               From =>  Data'First,
               To => Data'Last,
               Worker_Count => 4,
               Process  => Iterate'Access);
      end;

      Put_Line ("Parallel Elapsed=" &
            Duration'Image (To_Duration
                  (Clock - Start)));

   end Data_Block;

end Test_Loops;
```

When run on my machine AMD Quadcore with parameters 100_000 100_000, with full optimization turned on with -ftree-vectorize, I get.

```
Sequential Elapsed= 6.874298000
Parallel Elapsed= 6.287230000
With optimization turned off, I get
Sequential Elapsed= 32.428908000
Parallel Elapsed= 8.424717000
```

gcc with GNAT does a good job of optimization when its enabled, for these cases as shown, but the differences between optimization and using Paraffin can be more pronounced in other cases that are more complex, such as loops that involve reduction (e.g. calculating a sum)

Types and subtypes

From: Jean-Pierre Rosen
<rosen@adalog.fr>
Date: Wed, 13 Mar 2013 10:45:54 +0100
Subject: Re: Is this expected behavior or not
Newsgroups: comp.lang.ada

[…] In Ada, a type is a set of values. A subtype is the SAME TYPE, restricted to a subset of the values (the "constraint").

This feature does not exist in other OO languages, […] A subtype has the same operation as its base type, not copies of the operations as with inheritance.

Searching for NYU Ada/Ed Version 19.7 V-001

From: Nigel Williams
<nw@retrocomputingtasmania.com>
Date: Sat, 13 Apr 2013 16:56:19 -0700
Subject: searching for the first validated Ada compiler: NYU Ada/Ed
Newsgroups: comp.lang.ada

I'm part of small band of Ada enthusiasts and software preservationists who are keen to find and preserve the first validated Ada compiler (NYU Ada/Ed Version 19.7 V-001), implemented by NYU (nyu.edu) in the early 1980s and validated in 1983.

The validation report is referenced here:

http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA136759

NYU Ada/Ed was developed using the SETL programming language and it appears the first implementation was targeted at VAX/VMS. I hope to find both the Ada/Ed interpreter and the toolchain used to build it, at least minimally the relevant SETL implementation. It would also be important to find the V1.1 of the ACVC tests used for the validation as well.

I have contacted NYU and some of the original authors of NYU Ada/Ed but so far only later versions, particularly those that were re-built using C or SETL2 have so far been found.

We might be looking for old VAX backup tapes or perhaps someone kept printed copies of the source-code.

If anyone has leads and suggestions on what to look for and where to look, it would be appreciated.

TOYOTA ITC Japan selects SPARK Pro for High-Reliability research project

From: AdaCore Press Center
Date: Tue Apr 23 2013
Subject: TOYOTA ITC Japan Selects SPARK Pro Language and Toolset for High-Reliability Research Project
URL: http://www.adacore.com/press/toyota-itc-japan-selects-spark-pro-language-and-toolset-for-high-reliabilit/

SAN JOSE, Calif., NEW YORK and PARIS, April 23, 2013 – Design West Conference – AdaCore and Altran today announced TOYOTA InfoTechnology Center (ITC) Japan's selection of the SPARK language and SPARK Pro toolset for a high-reliability software research project. The goal of the project is to show that software requirements can be transformed into an implementation that can be proven to be free of run-time errors. This will have the key advantage of providing ultra-low-defect software for higher reliability in a vehicle component. An added benefit is the reduction of development and maintenance effort, since the formal approach being used can give mathematical assurance to a variety of correctness properties, reducing the need for certain types of testing and eliminating the need for post-deployment corrections.

The research project is taking a proven design and generating a fully assured code implementation, starting from a single vehicle system component. The aim is to use SPARK Pro technology to prove that the software can be produced free of run-time exceptions under all operating conditions, as a first step to composing larger ultra-low-defect systems. Alternative approaches using conventional software development methods have fundamental limitations. Testing can only provide evidence for a limited set of conditions, and static analysis performed on existing code to check for vulnerabilities, or other errors, does not address the underlying problem of preventing the errors in the first place. Using the SPARK language, toolset and methods solves this basic issue and will provide a clear competitive advantage for this component.

About SPARK

SPARK is a programming language that supports the precise specification of design or requirements in source code using a notation for formal contracts, including pre-conditions and post-conditions for subprograms, and inter-module information flow dependencies. The SPARK Pro toolset can then be used to verify that the software correctly implements the design, or meets its requirements, by verifying that the source code logic complies with the specified contracts.

SPARK can be used both to precisely express system requirements and to define an executable implementation, which can be formally shown to meet those requirements. Correctness can thus be demonstrated from the start, and maintained incrementally as the system evolves. This is a vastly different approach, and much more reliable, than developing a system and then using tests or static analysis to reduce the number of errors introduced in earlier life-cycle phases.

About SPARK Pro

SPARK Pro, a product jointly developed by Altran and AdaCore, provides a state-of-the-art language and toolset for engineering high-assurance software. It combines Altran's SPARK language and verification tools with AdaCore's GNAT Programming Studio (GPS) and GNATbench Integrated Development Environments. There are SPARK versions based on Ada 83, Ada 95, and Ada 2005, so all standard Ada compilers and tools work out-of-the-box with SPARK.

The SPARK Pro language and toolset is specifically designed for developing applications where correct operation is vital for safety or security. It offers static verification that is unrivalled in terms of its soundness (no "false negatives"), low false-alarm rate, depth and efficiency. The toolset generates evidence for correctness, including proofs of the absence of run-time errors that can be used to meet the requirements of safety and security certification schemes, such as ISO 26262, DO-178B, DO-178C and the Common Criteria. SPARK Pro is especially applicable in the context of the Formal Methods supplement to DO-178C.

About TOYOTA InfoTechnology Center Co., Ltd.

TOYOTA InfoTechnology Center Co., Ltd. provides cutting-edge technology and creates value with superior intelligence and greater innovation throughout the IT business related to automobiles. TOYOTA ITC as a whole has as its objective the development of advanced, world-class information technologies to meet market needs. This includes the research, development and evaluation of technologies, hardware and software research, analysis and planning of market and business models, and the management of intellectual property rights.

TOYOTA ITC has North America headquarters in Mountain View, CA and the main office in Tokyo, Japan. www.toyota-itc.com

[…]

Targeting GNAT to the TI MSP430

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Wed, 1 May 2013 23:37:10 +0000
Subject: Re: Extended Exceptions and other tweaks.
Newsgroups: comp.lang.ada

> I have borrowed the AVR-Ada RTS as a starting point for the TI MSP430 and it's looking good so far. I haven't tried exceptions yet though, but I'll report when I get that far.

Played with exceptions tonight; they look better than I expected and I think I can shed some light on Luke's experiments.

(NOTE: all the following apply with "pragma No_Exception_Propagation" in effect).

The a-except.ad[sb] from AVR-Ada _nearly_ worked for the MSP430 as well! The address clause on "procedure Reset" was the only point of failure so far, as it is not 0 on the MSP430 but 16#FF80# or other CPU-dependent value.

What's untested so far:

Actually running the executables : so far I have just examined the code emitted by the compiler

Raising exceptions as a result of runtime errors (overflow etc) instead of explicit Raise statements.

What works :

- Raising a standard exception.
  - Handling it locally
  - Handling it in the Last Chance Saloon (cough, Handler).
- Raising a user-defined exception, declared immediately visible to the subprogram which raises and handles it.
  - Handling it locally.
  - Handling it in the Last Chance handler.
- Raising a user-defined exception, declared elsewhere - even in a local package.
  - Handling it in the Last Chance handler.

What doesn't work :

Locally handling a user-defined exception from another package, via a selected name. The compiler ignores (deletes) any local handler and passes the exception (actually just plants a call to) the Last Chance handler. However, making the exception directly visible via a Use clause, it does work.

So the following ,

```
raise Timer_A.oops;   -- last chance handler!
and
use Timer_A;
raise oops;           -- can be locally handled
```

end up in different exception handlers!

I'm not clear on whether this is expected behaviour…

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 1 May 2013 20:06:20 -0500
Subject: Re: Extended Exceptions and other tweaks.
Newsgroups: comp.lang.ada

> I'm not clear on whether this is expected behaviour…

It's certainly not Ada behavior. But since you didn't show the local handler, there's a possibility that the problem was there, rather than in the exception raise. More likely, it is just some sort of compiler bug

Applying typestate analysis to Ada?

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 12 May 2013 00:06:25 +0300
Subject: Re: Seeking for papers about tagged types vs access to subprograms
Newsgroups: comp.lang.ada

[http://en.wikipedia.org/wiki/Typestate_analysis]

I haven't studied typestate analysis at all deeply. I had a look at the "Plaid" language referenced from the Wikipedia article, at http://www.cs.cmu.edu/~aldrich/plaid/, and was a bit surprised to understand that Ada subtypes come rather close to it, in particular subtypes of discriminated record types with variants. The value of the discriminant represents the state, with certain components of the record (= attributes of the object or type) existing or not existing depending on the variant selected by the discriminant.

For the Open-Close example:

```
type File_State is (Is_Closed, Is_Open);

type File_Object (State : File_State :=
        Is_Closed) is record
  case State is
  when Is_Closed => null;
  when Is_Open   => Handle :
        System.IO.File_Handle;
  end case;
end record;

subtype Closed_File is File_Object (State
        => Is_Closed);
subtype Open_File is File_Object (State
        => Is_Open  );
```

By using subtypes on formal parameters, we can indicate that the available operations on a File_Object depend on the actual subtype (i.e. the state), except for one deficiency, on which more below.

First, reading and writing is possible only for open files:

```
procedure Read (File : in Open_File; …)
procedure Write (File : in Open_File; …)
```

Second, a Closed file can be Opened, and an Open file can be Closed:

```
procedure Open
        (File : in out Closed_File);
procedure Close
        (File : in out Open_File  );
```

The problem here is that these operations should change the subtype of the "in out" parameter: Open changes the File from Closed_File to Open_File, and Close changes it from Open_File to Closed_File. However, Ada does not let us specify such changes, in the subprogram profile. (This can of course be specified with pre- and post-conditions, but I'm looking for a closer connection between the subprogram profile and the subtypes.)

Well, why is Ada limited in this way? There is no real reason why an "in out" parameter should have the same constraints on input and on output. So let us extend Ada to allow different "in" and "out" subtypes:

```
-- Extended Ada:
procedure Open
   (File : in Closed_File out Open_File ; …);
procedure Close
   (File : in Open_File  out Closed_File; …);
```

This gives us exactly the open/close typestate example. A compiler with strong value-range analysis should then be able to deduce, for example, that the "in" constraint check on a particular call of Open, Close, Read, or Write always succeeds (= typestate correctness) or may or must fail (various degrees of typestate incorrectness). I believe that many current compilers could do such value-range analysis, and thus Ada could support this kind of typestate concept with rather small language changes.

(You may ask: if an "in out" parameter has different "in" and "out" subtypes, what subtype is applied when the formal parameter is used or assigned within the subprogram body? I think the logical answer would be the "disjunctive subtype" that represents the union of the "in" and "out" subtypes, which unfortunately could be a subtype with "holes". But I think that would be manageable, since this disjunctive subtype would have at most two separate components, i.e. at most one hole.)

From: Shark8 <onewingedshark@gmail.com>
Date: Sat, 11 May 2013 16:19:06 -0700
Subject: Re: Seeking for papers about tagged types vs access to subprograms
Newsgroups: comp.lang.ada

> There is no real reason why an "in out" parameter should have the same constraints on input and on output.

What? Ada's not constrained like that, at least as you're implying:

Given the types you have […] the proper way to formulate _EXACTLY_ what you want is this:

```
procedure Open
        (File : in out File_Object; …)
 with Pre => File in Closed_File,
        Post => file in Open_File;
```

Or am I wrong?

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 12 May 2013 09:09:01 +0300
Subject: Re: Seeking for papers about tagged types vs access to subprograms
Newsgroups: comp.lang.ada

[…]

As I said, it can be expressed with pre- and post-conditions in this way, but then the parameter profile does not mention the subtypes and instead uses the unconstrained type (here File_Object). As I also said, I wanted the subtype changes to be visible in the subprogram profile, to show more clearly (or at least, more traditionally) how the availability of the subprogram depends on the state (i.e. the subtype) of the parameter, and how it affects the state.

To condense my points:

1. The typestate concept, as implemented in the Plaid language, seems (after my brief study of Plaid) to be implementable in Ada through discriminated records with variants.
2. The influence of the current typestate of an object, on the set of subprograms (operations) available for the object, can be represented as constraints on the "in" subtype of the object, and the typestate changes can be represented as the "out" subtype. In current Ada, of course, the subtype checks in principle occur at run-time, so typestate correctness is not checked at compile-time. Moreover, current Ada does not allow the formal subtype (as written in the profile) to be different for the "in" and "out" roles.
3. Those different "in" and "out" constraints can be implemented in Ada 2012 as pre/post-conditions, as you say.
4. A closer match to Plaid can be achieved if Ada is extended to allow different subtypes for the "in" and "out" roles of an "in out" parameter.

As the pre/post-condition feature of Ada 2012 becomes more familiar, perhaps the pre- and post-conditions will be seen as a more integrated part of the subprogram's profile, and there is no reason to consider changes to the formal subtypes allowed in the language (point 4).

On the other hand, it seems to me that there are other cases, not perhaps related to typestate, where it would be natural to specify different "in" and "out" subtypes

for an "in out" parameter. Something as simple as:

```
procedure Increment
        (Counter : in out Natural)
```

could become

```
-- Extended Ada:
procedure Increment ( Counter : in
        Natural range 0 ..Natural'Last - 1
        out Positive);
```

Of course, all such different in/out constraints can be expressed using the "heavy guns" of pre/post-conditions, so perhaps this suggestion is out of date after Ada 2012. Any compiler powerful enough to do useful typestate analysis based on the formal parameter subtypes is probably able to do the same analysis using the corresponding pre/post-conditions, at least when the conditions take the simple form "parameter in subtype".

It is interesting that Randy thinks his ideas regarding a future replacement for Ada resemble the typestate concept, but that the typestate concept as implemented in Plaid seems to be implementable in Ada 2012. Perhaps Randy's ideas go much further than this, however.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 13 May 2013 21:02:16 -0500
Subject: Re: Seeking for papers about tagged types vs access to subprograms
Newsgroups: comp.lang.ada

> […]

The main thing I was thinking about was some extension to the things that have to be compile-time analyzable (as Static Predicates are in Ada 2012). Probably the basis of them would remain subtypes and subprogram profiles (via preconditions and postconditions). For the Open example, that requires some way to encode the notion of "properties" in a statically understandable way. Perhaps you are right that discriminants would do the trick, but we'd want them to be "virtual" discriminants without any runtime cost.

I agree that you can get the effect of typestate analysis in Ada 2012 using discriminants, predicates, and pre/postconditions, but those would be checked at runtime. The key here for me is to require static detection of these errors, even when variables and unconstrained formal parameters are involved.

Anyway, I'm just musing here as opposed to having fully worked out ideas. So I could in fact be going down the wrong path.

The point in having an object for a closed file

From: Jeffrey Carter <jrcarter@acm.org>
Date: Sun, 12 May 2013 11:56:50 -0700
Subject: Re: Seeking for papers about tagged types vs access to subprograms
Newsgroups: comp.lang.ada

> […]

I'm not sure I see the point in having an object for a closed file, other than the requirements of low-level languages in which such things were 1st implemented. Why not something like

```
type File_Info (<>) is tagged limited
        private;

function Open  (Name : ...; ...)
        return File_Info;
function Create (Name : …; …)
        return File_Info;

function Read  (File : in out File_Info)
        return …;
procedure Write (File : in out File_Info;
        Item : in …);

declare
  File : File_Info := Open ("junk", …);
begin
  Data := Read (File);
  …
end;
```

A File_Info must be opened or created when declared, and is closed when it's finalized.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Sun, 12 May 2013 18:15:24 -0400
Subject: Re: Seeking for papers about tagged types vs access to subprograms
Newsgroups: comp.lang.ada

> I'm not sure I see the point in having an object for a closed file,

Yeah, I was about to post basically the same thing. A closed file is pretty useless. It's like an uninitialized variable -- you can't do anything with it.

[…]

> type File_Info (<>) is tagged limited private;

Yes, but I would have separate types for Input_File and Output_File. Possibly another type for the rare case when you want to read and write to/from the same file handle.

> function Open (Name : …; …) return File_Info;

> function Create (Name : …; …) return File_Info;

Note that these are build-in-place functions. You can't call them as the right-hand side of an assignment statement.

> […]

You could use it as above, or like this:

```
Grind_Upon_File(Open("junk", ...));
```

Or like this:

```
X := new File_Info'(Open(...));
```

But you couldn't use it like this:

```
File : File_Info; -- Illegal!
… -- some code that computes File_Name
File := Open (File_Name); -- Illegal!
```

which is a limitation, compared to the current design of Text_IO and friends.

Conference Calendar

Dirk Craeynest
KU Leuven. Email: Dirk.Craeynest@cs.kuleuven.be

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

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

2013

☺ July 01-02 **International Symposium on High-Level Parallel Programming and applications** (HLPP'2013), Paris, France.

July 01-03 18th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2013), Canterbury, Kent, UK.

July 01-03 7th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2013), Birmingham, UK. Topics include: the theoretical aspects of model driven software engineering, component based software engineering, software security, reliability, and verification, embedded and real time software systems, aspect and object oriented software design, reverse engineering, etc.

☺ July 01-05 27th **European Conference on Object-Oriented Programming** (ECOOP'2013), Montpellier, France. Topics include: all areas of object technology and related software development technologies, such as aspects, components, modularity, concurrent and parallel systems, distributed computing, programming environments, versioning, refactoring, software evolution, language definition and design, language implementation, compiler construction, design methods and design patterns, real-time systems, security, specification, verification, type systems, etc.

- July 01 **Workshop on MechAnisms for SPEcialization, Generalization and inHerItance** (MASPEGHI'2013). Topics include: the design of inheritance-related reuse mechanisms, including their dynamic semantics, static analysis, permissions and visibility; software engineering issues, including metrics, interactions with methodologies, and consequences for quality parameters such as maintainability and comprehensibility.
- ☺ July 01 2nd **International Workshop on Combined Object-Oriented Modeling and Programming Languages** (COOMPL'2013). Topics include: differences and similarities between modeling and programming, modeling constructs not supported by programming languages, programming constructs not supported by modeling languages, support for functional and constraint programming in modeling, support for concurrent / distributed modeling and programming, implementation techniques, etc.
- ☺ July 01-02 **Doctoral Symposium**. Topics include: languages, modelling, processes, environments and tools, methods and programming/modelling paradigms, execution, concurrent parallel and distributed systems, evolution, analysis validation and verification.
- ☺ July 02 8th **Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems** (ICOOOLPS'2013). Topics include: efficient implementation and compilation of OO languages in various application domains ranging from embedded and real-time systems to desktop systems.

July 03-07 8th **International Conference on Evaluation of Novel Approaches to Software Engineering** (ENASE'2013), Angers, France. Topics include: emerging as well as established SE methods, practices, architectures, technologies and tools; software process improvement, model-driven engineering, application integration technologies, software quality management, software change and configuration management, geographically distributed software development environments, formal methods,

component-based software engineering and commercial-off-the-shelf (COTS) systems, software and systems development methodologies, etc.

☺ July 12-14 **GNU Tools Cauldron** 2013, Mountain View, California, USA. Topics include: gathering of GNU tools developers, to discuss current/future work, coordinate efforts, exchange reports on ongoing efforts, discuss development plans for the next 12 months, developer tutorials and any other related discussions.

☺ July 16-18 11th IEEE **International Symposium on Parallel and Distributed Processing with Applications** (ISPA'2013), Melbourne, Australia. 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 17-19 18th IEEE **International Conference on the Engineering of Complex Computer Systems** (ICECCS'2013), Singapore. Topics include: verification and validation, security of complex systems, 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.

July 23-25 25th ACM **Symposium on Parallelism in Algorithms and Architectures** (SPAA'2013), Montreal, Quebec, Canada. Topics include: parallel and distributed algorithms; multi-core architectures; compilers and tools for concurrent programming; synergy of parallelism in algorithms, programming, and architecture; etc.

July 29-30 13th **International Conference on Quality Software** (QSIC'2013), Nanjing, China. Theme: "Quality of Evolving Software". Topics include: dynamic analysis, software quality, inspection, fault localization, code review, formal methods, static analysis, proof-based systems, verification techniques combining proofs and tests, testing in multi-core environments, etc.

August 18-26 9th **Joint European Software Engineering Conference** and ACM SIGSOFT **Symposium on the Foundations of Software Engineering** (ESEC/FSE'2013), Saint Petersburg, Russia. Topics include: components, development environments and tools, distributed software, embedded and real-time software, maintenance and evolution, model-driven software engineering, parallel and concurrent software, reverse engineering, software architecture, validation, verification, and testing, etc.

☺ Aug 19-20 **International Conference on Multicore Software Engineering, Performance, and Tools** (MUSEPAT'2013). Topics include: software engineering for multicore systems; specification, modeling and design; programing models, languages, compiler techniques and development tools; verification, testing, analysis; debugging, performance tuning, and security testing; software maintenance and evolution; multicore software issues in scientific computing, embedded and mobile systems; energy-efficient computing; experience reports.

☺ August 19-21 19th IEEE **International Conference on Embedded and Real-Time Computing Systems and Applications** (RTCSA'2013), Taipei, Taiwan. Topics include: embedded system design practices, software and compiler issues for heterogeneous multi-core embedded platform, real-time scheduling, timing analysis, programming languages and run-time systems, middleware systems, design and analysis tools, case studies and applications, etc.

☺ August 26-30 19th **International European Conference on Parallel and Distributed Computing** (Euro-Par'2013), Aachen, Germany. Topics include: all aspects of parallel and distributed computing, such as support tools and environments, scheduling, high-performance compilers, distributed systems and algorithms, parallel and distributed programming, multicore and manycore programming, theory and algorithms for parallel computation, etc.

September 04-06 39th Euromicro **Conference on Software Engineering and Advanced Applications** (SEAA'2013), Santander, Spain. Topics include: information technology for software-intensive systems.

September 08-11 10th **International Conference on Parallel Processing and Applied Mathematics** (PPAM'2013), Warsaw, Poland. Topics include: multi-core and many-core parallel computing; parallel/distributed algorithms: numerical and non-numerical; scheduling, mapping, load balancing; parallel/distributed programming; tools and environments for parallel/distributed computing; security and dependability in

parallel/distributed environments; applications of parallel/distributed computing; etc. Event also includes: workshop on Language-Based Parallel Programming Models.

☺ Sep 08-11 5th **Workshop on Language-Based Parallel Programming Models** (WLPP'2013). Topics include: Language and library implementations; Proposals for, and evaluation of, language extensions; Applications development experiences; Comparisons between programming models; Compiler Implementation and Optimization; etc.

September 08-11 FedCSIS2013 - 4th **Workshop on Advances in Programming Languages** (WAPL'2013), Kraków, Poland. Topics include: compiling techniques, domain-specific languages, formal semantics and syntax, generative and generic programming, languages and tools for trustworthy computing, language concepts, design and implementation, model-driven engineering languages and systems, practical experiences with programming languages, program analysis, optimization and verification, programming tools and environments, proof theory for programs, specification languages, type systems, etc.

☺ September 10-13 **International Conference on Parallel Computing** 2013 (ParCo'2013), München, Germany. Topics include: all aspects of parallel computing, including applications, hardware and software technologies as well as languages and development environments, in particular Parallel programming languages, compilers, and environments; Tools and techniques for generating reliable and efficient parallel code; Best practices of parallel computing on multicore, manycore, and stream processors; etc. Deadline for submissions: July 31, 2013 (full papers).

September 11-13 13th **Workshop on Automated Verification of Critical Systems** (AVoCS'2013), Guilford, Surrey, UK. Topics include: specification and refinement, verification of software and hardware, real-time systems, dependable systems, verified system development, industrial applications, etc. Deadline for submissions: July 19, 2013 (short papers). Deadline for registration: July 30, 2013.

September 22-23 13th IEEE **International Working Conference on Source Code Analysis and Manipulation** (SCAM'2013), Eindhoven, the Netherlands. Topics include: program transformation and refactoring, static and dynamic analysis, source level software metrics, security and vulnerability analysis, source level verification, program comprehension, abstract interpretation, etc.

September 22-24 12th **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SoMeT'2013), Budapest, Hungary. 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.

September 22-28 29th IEEE **International Conference on Software Maintenance** (ICSM'2013), Eindhoven, the Netherlands. Topics include: software repository analysis and mining; reverse engineering, re-engineering and migration; software refactoring, restructuring and renovation; software and system comprehension; maintenance-related testing (e.g., regression testing); maintenance and evolution processes; software quality improvement; etc.

☺ Sep 23-24 18th **International Workshop on Formal Methods for Industrial Critical Systems** (FMICS'2013), Madrid, Spain. 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.

September 23-27 11th **International Conference on Software Engineering and Formal Methods** (SEFM'2013), Madrid, Spain. 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.

☺ Sep 29 - Oct 04 CBSoft2013 - 17th **Brazilian Symposium on Programming Languages** (SBLP'2013), Brasília, Distrito Federal, Brazil. Topics include: the fundamental principles and innovations in the design and implementation of programming languages and systems; programming paradigms and styles, including object-oriented, real-time, multithreaded, parallel, and distributed programming; program analysis and verification, including type systems, static analysis and abstract interpretation; programming language design and implementation, including new programming models, programming language environments, compilation and interpretation techniques; etc.

Sep 30 - Oct 03 32nd **International Symposium on Reliable Distributed Systems** (SRDS'2013), Braga, Portugal. Topics include: distributed objects and middleware systems, enabling technologies for dependable applications, formal methods and foundations for dependable distributed computing, analytical or experimental evaluations of dependable distributed systems, secure and trusted systems, high-assurance and safety-critical system design and evaluation, etc.

☺ Sep 30- Oct 04 12th **International Conference on Parallel Computing Technologies** (PaCT'2013), Saint-Petersburg, Russia. Topics include: new developments, applications, and trends in parallel computing technologies; all technological aspects of the applications of parallel computer systems; high level parallel programming languages and systems; methods and tools for parallel solution of large-scale problems; languages, environments and software tools supporting parallel processing; teaching parallel processing; etc.

☺ Sep 7 6th **International Workshop on Multi/many-Core Computing Systems** (MuCoCoS'2013). Topics include: portable programming models, languages and compilation techniques; case studies highlighting performance portability and tuning; etc.

☺ October 03 ICPP2013 - **International Workshop on Embedded Multicore Systems** (EMS'2013), Lyon, France. Topics include: programming models for embedded multicore systems; software for Multicore, GPU, and embedded architectures; real-time system designs for embedded multicore environments; applications for automobile electronics of multicore designs; compiler for worst-case execution time analysis; formal method for embedded systems; etc.

October 10-11 7th **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2013), Baltimore, Maryland, USA. Topics include: qualitative methods; replication of empirical studies; 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.

October 14-17 20th **Working Conference on Reverse Engineering** (WCRE'2013), Koblenz, Germany. Topics include: program comprehension, reengineering to distributed systems, mining software repositories, software architecture recovery, empirical studies in reverse engineering, program analysis and slicing, re-documenting legacy systems, reengineering patterns, program transformation and refactoring, reverse engineering tool support, etc. Deadline for submissions: July 1, 2013 (tool demonstrations), July 10, 2013 (workshops).

October 26-28 6th **International Conference on Software Language Engineering** (SLE'2013), Indiana, Indianapolis, USA. 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 7, 2013 (abstracts), June 14, 2013 (full papers).

☺ October 26-31 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2013), Indiana, Indianapolis, USA. Deadline for submissions: July 19 - September 6, 2013 (workshop papers).

☺ Oct 26 - 31 28th **Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2013). Topics include: any aspect of programming, systems, languages, and applications; any aspect of software development, including requirements, modeling, prototyping, design, implementation, generation, analysis,

verification, testing, evaluation, maintenance, reuse, replacement, and retirement of software systems; large-scale software repositories; tools (such as new languages, program analyses, or runtime systems) or techniques (such as new methodologies, design processes, code organization approaches, and management techniques) that go beyond objects in interesting ways; etc.

Oct 27 - Nov 01 8th **International Conference on Software Engineering Advances** (ICSEA'2013), Venice, Italy. 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; improving productivity in research on software engineering; etc.

Oct 29 - Nov 01 15th **International Conference on Formal Engineering Methods** (ICFEM'2013), Queenstown, New Zealand. Topics include: abstraction and refinement; program analysis; software verification; formal methods for software safety, security, reliability and dependability; tool development, integration and experiments involving verified systems; formal methods used in certifying products under international standards; formal model-based development and code generation; etc.

November 04-07 24th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2013), Pasadena, CA, USA.

♦ Nov 10-14 ACM SIGAda **Annual International Conference on High Integrity Language Technology** (HILT'2013), Pittsburgh, Pennsylvania, USA. Deadline for submissions: August 1, 2013 (industrial presentations).

November 17-22 26th **International Conference for High Performance Computing, Networking, Storage and Analysis** (SC'2013), Denver, Colorado, USA. Topics include: applications, programming systems (technologies that support parallel programming, such as compiler analysis and optimization, parallel programming languages and notations, programming models, runtime systems, tools, software engineering for parallel programming, solutions for parallel programming challenges, ...), state of the practice, etc. Deadline for submissions: July 31, 2013 (posters).

December 02-05 20th **Asia-Pacific Software Engineering Conference** (APSEC'2013), Bangkok, Thailand. Topics include: software engineering methodologies; software analysis and understanding; software testing, verification and validation; software maintenance and evolution; software quality and measurement; software process and standards; software security, reliability and privacy; software engineering environments and tools; software engineering education; distributed and parallel software systems; embedded and real-time software systems; formal methods in software engineering; etc. Deadline for submissions: July 30, 2013 (industry track papers, postgraduate symposium papers, tutorials).

December 09-11 11th **Asian Symposium on Programming Languages and Systems** (APLAS'2013), Melbourne, Australia. Topics include: foundational and practical issues in programming languages and systems, such as semantics, design of languages and type systems, domain-specific languages, compilers, interpreters, abstract machines, program analysis, verification, model-checking, software security, concurrency and parallelism, tools and environments for programming and implementation, etc.

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

☺ December 15-18 19th IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2013), Seoul, Korea. Topics include: parallel and distributed applications and algorithms, multi-core and multithreaded architectures, security and privacy, dependable and trustworthy computing and systems, real-time systems, cyber-physical systems, embedded systems, etc.

December 18-21 20th IEEE **International Conference on High Performance Computing** (HiPC'2013), Hyderabad, India. Topics include: parallel and distributed algorithms / applications, parallel languages and programming environments, hybrid parallel programming with GPUs, scheduling, resilient/fault-tolerant algorithms and systems, scientific/engineering/commercial applications, compiler technologies for high-performance computing, software support, etc.

2014

January 09-11 — 15th IEEE **International Symposium on High Assurance Systems Engineering** (HASE'2014), Miami, Florida, USA. Topics include: tools and techniques used to design and construct systems that, in addition to meeting their functional objectives, are safe, secure, and reliable. Deadline for submissions: August 1, 2013 (papers).

February 12-14 — 22nd Euromicro **International Conference on Parallel, Distributed and Network-Based Computing** (PDP'2014), Turin, Italy. Topics include: embedded parallel and distributed systems, multi- and many-core systems, programming languages and environments, runtime support systems, simulation of parallel and distributed systems, dependability and survivability, real-time distributed applications, etc. Deadline for submissions: July 31, 2013 (full papers).

March 24-28 — 29th ACM **Symposium on Applied Computing** (SAC'2014), Gyeongju, Korea.

- ☺ Mar 24-28 **Track on Programming Languages** (PL'2014). Topics include: compiling techniques, domain-specific languages, formal semantics and syntax, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, programming languages from all paradigms, etc.
- March 24-28 **Track on Software Verification and Testing** (SVT'2014). Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, etc. Deadline for submissions: September 13, 2013.

April 05-13 — **European Joint Conferences on Theory and Practice of Software** (ETAPS'2014), Grenoble, France. Events include: CC, International Conference on Compiler Construction; ESOP, European Symposium on Programming; FASE, Fundamental Approaches to Software Engineering; FOSSACS, Foundations of Software Science and Computation Structures; POST, Principles of Security and Trust; TACAS, Tools and Algorithms for the Construction and Analysis of Systems.

April 22-26 — 13th **International Conference on Modularity** (Modularity'2013), Lugano, Switzerland. Topics include: varieties of modularity (generative programming, aspect orientation, software product lines, components; ...); programming languages (support for modularity related abstraction in: language design; verification, contracts, and static program analysis; compilation, interpretation, and runtime support; formal languages; ...); software design and engineering (evolution, empirical studies of existing software, economics, testing and verification, composition, methodologies, ...); tools (refactoring, evolution and reverse engineering, support for new language constructs, ...); applications (distributed and concurrent systems, middleware, cyber-physical systems, ...); complex systems; etc. Deadline for submissions: July 25, 2013 (round 1), October 13, 2013 (round 2).

♦ June 23-27 — 19th **International Conference on Reliable Software Technologies - Ada-Europe'2014**, Paris, France. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN (requests pending).

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

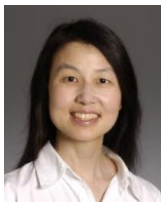
ACM SIGAda Annual International Conference
High Integrity Language Technology HILT 2013
Call for Technical Contributions

Developing and Certifying Critical Software

Pittsburgh, Pennsylvania, USA
November 10-14, 2013

Sponsored by ACM SIGAda in cooperation with SIGPLAN, SIGAPP, SIGCSE, SIGBED, SIGCAS, SIGSOFT, Ada-Europe and the Ada Resource Association
Contact: SIGAda.HILT2013 at acm.org www.sigada.org/conf/hilt2013

KEYNOTE & INVITED SPEAKERS

Edmund M. Clarke (ACM Turning Award 2007 and Professor of Electrical and Computer Engineering, Carnegie Mellon University), Jeannette Wing (Vice President and Head Microsoft Research International) and John Goodenough (Software Engineering Institute). Special session on Model-Based Engineering with invited talk by Michael Whalen of University of Minnesota.

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 2013 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**. We are soliciting technical papers, experience reports, 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 mare than 100,000 members, and the wider 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 (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: **Except for Tutorial proposals use www.easychair.org/conferences/?conf=hilt2013**

Submission	*Deadline*	*Use Easy Chair Link Above*
Technical articles, extended abstracts, experience reports, panel session proposals, or workshop proposals	**June 29, 2013**	For more info contact: **Tucker Taft**, Program Chair `taft@adacore.com`
Industrial presentation proposals	**August 1, 2013** (overview) **September 30, 2013** (abstract)	
Send **Tutorial** proposals to	**June 29, 2013**	**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 **Greg Gicca** (`greggicca@gmail.com`) to learn the benefits of becoming a sponsor and/or exhibitor at HILT 2013.

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.HILT2013@acm.org or Conference Chair (**Jeff Boleng**, jlboleng@SEI.CMU.EDU), Program Chair (**Tucker Taft**, taft@adacore.com), SIGAda's Vice-Chair for Meetings and Conferences (**Alok Srivastava**, alok.srivastava@tasc.com), or SIGAda's Chair (**Ricky E. Sward**, rsward@mitre.org).

First Call for Papers

19th International Conference on Reliable Software Technologies – Ada-Europe 2014

23-27 June 2014, Paris, France

http://www.ada-europe.org/conference2014

General Chair

Jean-Pierre Rosen
Adalog
rosen@adalog.fr

Program co-Chairs

Laurent George
LIGM/UPEMLV - ECE Paris
lgeorge@ieee.org

Tullio Vardanega
University of Padova
tullio.vardanega@unipd.it

Industrial Chair

Jørgen Bundgaard
Rambøll Denmark A/S
jogb@ramboll.dk

Tutorial co-Chairs

Liliana Cucu
INRIA
Liliana.Cucu@inria.fr

Albert Llemosí
Universitat de les Illes Balears
albert.llemosi@uib.cat

Exhibition Chair

To be appointed

Publicity co-Chairs

Jamie Ayre
AdaCore
ayre@adacore.com

Dirk Craeynest
Ada-Belgium & KU Leuven
Dirk.Craeynest@cs.kuleuven.be

Local Chair

Magali Munos
ECE
munos@ece.fr

In cooperation requested with ACM SIGAda, SIGBED, SIGPLAN

acm In-Cooperation

General Information

The 19th International Conference on Reliable Software Technologies – Ada-Europe 2014 will take place in Paris, France. As per its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical and industrial programs, along with tutorials and workshops on Monday and Friday.

Schedule

8 December 2013	Submission of regular papers, tutorial and workshop proposals
19 January 2014	Submission of industrial presentation proposals
16 February 2014	Notification of acceptance to all authors
16 March 2014	Camera-ready version of regular papers required
18 May 2014	Industrial presentations, tutorial and workshop material required

Topics

The conference has over the years become a leading international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will allow ample time for keynotes, Q&A sessions and discussions, and social events. Participants include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies.

Topics of interest to this edition of the conference include but are not limited to:

- **Multicore and Manycore Programming**: Predictable Programming Approaches for Multicore and Manycore Systems, Parallel Programming Models, Scheduling Analysis Techniques.
- **Real-Time and Embedded Systems**: Real-Time Scheduling, Design Methods and Techniques, Architecture Modelling, HW/SW Co-Design, Reliability and Performance Analysis.
- **Theory and Practice of High-Integrity Systems**: Challenges from Mixed-Criticality Systems; Medium to Large-Scale Distribution, Fault Tolerance, Security, Reliability, Trust and Safety, Languages Vulnerabilities.
- **Software Architectures**: Design Patterns, Frameworks, Architecture-Centred Development, Component-based Design and Development.
- **Methods and Techniques for Software Development and Maintenance**: Requirements Engineering, Model-driven Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- **Enabling Technologies**: Compilers, Support Tools (Analysis, Code/Document Generation, Profiling), Run-time Systems and Libraries.
- **Software Quality:** Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Cloud Environments, Smart Energy systems, Serious Games, etc.
- **Experience Reports in Reliable System Development**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **Experiences with Ada and its Future**: Reviews of the Ada 2012 new language features; implementation and use issues; positioning in the market and in the software engineering curriculum; lessons learned on Ada Education and Training Activities with bearing on any of the conference topics.

Program Committee

Mario Aldea, Universidad de Cantabria, Spain
Ted Baker, US National Science Foundation, USA
Johann Blieberger, Technische Universität Wien, Austria
Bernd Burgstaller, Yonsei University, Korea
Maryline Chetto, University of Nantes, France
Liliana Cucu, INRIA, France
Christian Fraboul, ENSEEIHT, France
Laurent George, ECE Paris, France
Xavier Grave, CNRS, France
Emmanuel Grolleau, ENSMA, France
Jérôme Hugues, ISAE, France
Albert Llemosí, Universitat de les Illes Balears, Spain
Kristina Lundqvist, Mälardalen University, Sweden
Franco Mazzanti, ISTI-CNR, Italy
John McCormick, University of Northern Iowa, USA
Stephen Michell, Maurya Software, Canada
Laurent Pautet, Telecom ParisTech, France
Luís Miguel Pinho, CISTER/ISEP, Portugal
Erhard Plödereder, Universität Stuttgart, Germany
Juan A. de la Puente, Universidad Politécnica de Madrid, Spain
Jorge Real, Universitat Politècnica de València, Spain
José Ruiz, AdaCore, France
Sergio Sáez, Universitat Politècnica de València, Spain
Amund Skavhaug, NTNU, Norway
Yves Sorel, INRIA, France
Tucker Taft, AdaCore, USA
Theodor Tempelmeier, University of Applied Sciences, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Tullio Vardanega, University of Padova, Italy
Juan Zamorano, Universidad Politécnica de Madrid, Spain

Industrial Committee

Jacob Sparre Andersen, JSA Consulting, Denmark
Roger Brandt, Telia, Sweden
Ian Broster, Rapita Systems, UK
Jørgen Bundgaard, Rambøll, DK
Dirk Craeynest, Ada-Belgium & KU Leuven, Belgium
Peter Dencker, ETAS, Germany
Ismael Lafoz, Airbus, Spain
Maria del Carmen Lomba Sorrondegui, GMV, Spain
Ahlan Marriott, White Elephant, CH
Robin Messer, Altran-Praxis, UK
Quentin Ochem, AdaCore, France
Steen Palm, Terma, Denmark
Paolo Panaroni, Intecs, Italy
Paul Parkinson, Wind River, UK
Ana Rodriguez, Silver-Atena, Spain
Jean-Pierre Rosen, Adalog, France
Alok Srivastava, TASC, USA
Claus Stellwag, Elektrobit, Germany
Jean-Loup Terraillon, European Space Agency, Netherlands
Rod White, MBDA, UK

Call for Regular Papers

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall exceed 14 LNCS-style pages in length. Authors shall submit their work via EasyChair following the relevant link on the conference web site. The format for submission is solely PDF.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the start of the conference. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by March 16, 2014. For format and style guidelines authors should refer to *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The CORE ranking (dated 2008) has the conference in class A. The CiteSeerX Venue Impact Factor had it in the top quarter. Microsoft Academic Search has it in the top third for conferences on programming languages by number of citations in the last 10 years. The conference is listed in DBLP, SCOPUS and Web of Science Conference Proceedings Citation index, among others.

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Industrial Presentations

The conference seeks industrial presentations which deliver value and insight but may not fit the selection process for regular papers. Authors are invited to submit a presentation outline of exactly 1 page in length by January 19, 2014. Submissions shall be made via EasyChair following the relevant link on the conference web site. The *Industrial Committee* will review the submissions and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by May 18, 2014, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the *Industrial Chair* directly.

Call for Tutorials

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

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference week. Workshop proposals should be submitted to the *General Chair*. The workshop organizer shall also commit to preparing proceedings for timely publication in the Ada User Journal.

Call for Exhibitors

The commercial exhibition will span the three days of the main conference. Vendors and providers of software products and services should contact a *Conference Co-Chair* for information and for allowing suitable planning of the exhibition space and time.

Grant for Reduced Student Fees

A limited number of sponsored grants for reduced fees is expected to be available for students who would like to attend the conference or tutorials. Contact the *General Chair* for details.

Rationale for Ada 2012: 6a Containers

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 improvements to the predefined container library in Ada 2012.

Keywords: rationale, Ada 2012.

1 Overview of changes

The WG9 guidance document [1] specifically says that attention should be paid to

> improving the use and functionality of the predefined containers.

The predefined containers were introduced in Ada 2005 and experience with their use revealed a number of areas where they could be improved.

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

1 Bounded containers and other container issues

69 Holder container

136 Multiway tree container

139 Syntactic sugar for access, containers & iterators

159 Queue containers

184 Compatibility of streaming of containers

212 Accessors and iterators for Ada.Containers

251 Problems with queue containers

These changes can be grouped as follows.

The existing containers are unbounded and generally require dynamic storage management to be performed behind the scenes. However, for high-integrity systems, such dynamic management is often unacceptable. Accordingly, bounded versions of all the existing containers are added (1).

A number of facilities are added to make important operations on containers more elegant. These are the updating of individual elements of a container and iteration over a container (139, 212).

Ada 2005 introduced containers for the manipulation of lists and it was expected that this would provide a basis for manipulating trees. However, this proved not to be the case, so specific containers are added for the manipulation of multiway trees (136). There are versions for unbounded indefinite and unbounded definite trees and for bounded definite trees.

A further new kind of container is for single indefinite objects and is known as the holder container (69).

A range of containers are added for manipulating queues with defined behaviour regarding multiple task access to the queues (159, 251).

The Ada 2005 container library also introduced sorting procedures for constrained and unconstrained arrays. An additional more general sorting mechanism is added in Ada 2012 (1).

Finally, an oversight regarding the streaming of containers is corrected (184).

2 Bounded and unbounded containers

It is perhaps worth starting this discussion by summarizing the containers introduced in Ada 2005. First, there is a parent package Ada.Containers which simply declares the types Hash_Type and Count_Type.

Then there are six containers for definite objects, namely (abbreviating the prefix Ada.Containers to just A.C)

```
A.C.Vectors
A.C.Doubly_Linked_Lists
A.C.Hashed_Maps
A.C.Ordered_Maps
A.C.Hashed_Sets
A.C.Ordered_Sets
```

The declarations of these six containers all start with

```
generic
   ...
   type Element_Type is private;
   ...
package Ada.Containers.XXX...
```

and we see that the type Element_Type has to be definite. There are also containers for the manipulation of indefinite types whose names are

```
A.C.Indefinite_Vectors
A.C.Indefinite_Doubly_Linked_Lists
A.C.Indefinite_Hashed_Maps
A.C.Indefinite_Ordered_Maps
A.C.Indefinite_Hashed_Sets
A.C.Indefinite_Ordered_Sets
```

and these are very similar to the definite containers except that the formal type Element_Type is now declared as

```
   type Element_Type(<>) is private;
```

so that the actual type can be indefinite such as String.

Finally, there are two generic packages for sorting arrays namely

A.C.Generic_Array_Sort
A.C.Generic_Constrained_Array_Sort

which apply to unconstrained and constrained arrays respectively.

The first change in Ada 2012 is that the parent package Ada.Containers now includes the declaration of the exception Capacity_Error so that it becomes

```
package Ada.Containers is
  pragma Pure(Containers);

  type Hash_Type is mod implementation-defined;
  type Count_Type is range 0 .. implementation-defined;
  Capacity_Error: exception;

end Ada.Containers;
```

The names of the new containers with bounded storage capacity are

A.C.Bounded_Vectors
A.C.Bounded_Doubly_Linked_Lists
A.C.Bounded_Hashed_Maps
A.C.Bounded_Ordered_Maps
A.C.Bounded_Hashed_Sets
A.C.Bounded_Ordered_Sets

The facilities of the bounded containers are almost identical to those of the original unbounded ones so that converting a program using one form to the other is relatively straightforward. The key point of the bounded ones is that storage management is guaranteed (implementation advice really) not to use features such as pointers or dynamic allocation and therefore can be used in high-integrity or safety-critical applications.

The major differences between the packages naturally concern their capacity. In the case of the bounded packages the types such as Vector have discriminants thus

```
type Vector(Capacity: Count_Type) is tagged private;
```

whereas in the original packages the type Vector is simply

```
type Vector is tagged private;
```

The other types in the bounded packages are

```
type List(Capacity: Count_Type) is tagged private;

type Map(Capacity: Count_Type;
         Modulus: Hash_Type) is tagged private;

type Map(Capacity: Count_Type) is tagged private;

type Set(Capacity: Count_Type;
         Modulus: Hash_Type) is tagged private;

type Set(Capacity: Count_Type) is tagged private;
```

Note that the types for hashed maps and sets have an extra discriminant to set the modulus; this will be explained in a moment.

Remember that the types Count_Type and Hash_Type are declared in the parent package Ada.Containers shown above.

When a bounded container is declared, its capacity is set once and for all by the discriminant and cannot be changed. If we subsequently add more elements to the container than it can hold then the exception Capacity_Error is raised.

If we are using a bounded container and want to make it larger then we cannot. But what we can do is create another bounded container with a larger capacity and copy the values from the old container to the new one. Remember that we can check the number of items in a container by calling the function Length.

So we might have a sequence such as

```
My_List: List(100);
...                                  -- use my list
if Length(My_List) > 90 then         -- Gosh, nearly full
...
  declare
    My_Big_List: List := Copy(My_List, 200);
  begin
    ...
```

The specification of the function Copy is

```
function Copy(Source: List;
              Capacity: Count_Type := 0) return List;
```

If the parameter Capacity is not specified (or is given as zero) then the capacity of the copied list is the same as the length of Source.

If the given value of Capacity is larger than (or equal to) the length of the Source (as in our example) then the returned list has this capacity and the various elements are copied. If we foolishly supply a value which is less than the length of Source then Capacity_Error is naturally raised. Remember that a discriminant can be set by an initial value.

Note that if we write

```
declare
  My_Copied_List: List := My_List;
begin
```

then My_Copied_List will have the same capacity as My_List because discriminants are copied as well as the contents.

In order to make it easier to move from the bounded form to the unbounded form, a function Copy is added to the unbounded containers as well although it does not need a parameter Capacity in the case of lists and ordered maps and sets. So in the case of the list container it is simply

```
function Copy(Source: List) return List;   -- unbounded
```

Similar unification between bounded and unbounded forms occurs with assignment. In Ada 2005, if we have two lists L and M, then we can simply write

```
L := M;
```

and the whole structure is copied (including all its management stuff). Note that this will almost certainly require that the value of L be finalized which might be a nuisance. Such an assignment with discriminated types needs to check the discriminants as well (and raises Constraint_Error if they are different). This is a nuisance because although the capacities might not be the same, the destination L might have plenty of room for the actual elements in the source M.

This is all rather bothersome and so procedures Assign are added to both unbounded and bounded containers which simply copy the element values. Thus in both case we have

```
procedure Assign(Target: in out List; Source: in List);
```

In the bounded case, if the length of Source is greater than the capacity of Target, then Capacity_Error is raised. In the unbounded case, the structure is automatically extended.

It might be recalled that in Ada 2005, lists and ordered maps and sets do not explicitly have a notion of capacity. It is in their very nature that they automatically extend as required. However, in the case of vectors and hashed maps and sets (which have a notion of indexing) taking a purely automatic approach could lead to lots of extensions and copying so the notion of capacity was introduced. The capacity can be set by calling

```
procedure Reserve_Capacity(Container: in out Vector;
                           Capacity: in Count_Type);
```

and the current value of the capacity can be ascertained by calling

```
function Capacity(Container: Vector) return Count_Type;
```

which naturally returns the current capacity. Note that Length(V) cannot exceed Capacity(V) but might be much less.

If we add items to a vector whose length and capacity are the same then no harm is done. The capacity will be expanded automatically by effectively calling Reserve_Capacity internally. So the user does not need to set the capacity although not doing so might result in poorer performance.

The above refers to the existing unbounded forms and is unchanged in Ada 2012. For uniformity the new bounded forms for vectors and hashed maps and sets also declare a procedure Reserve_Capacity. However, since the capacity cannot be changed for the bounded forms it simply checks that the value of the parameter Capacity does not exceed the actual capacity of the container; if it does then Capacity_Error is raised and otherwise it does nothing. There is of course also a function Capacity for bounded vectors and hashed maps and sets which simply returns the fixed value of the capacity.

Many operations add elements to a container. For unbounded containers, they are automatically extended as necessary as just explained. For the bounded containers, if an operation would cause the capacity to be exceeded then Capacity_Error is raised.

There are a number of other differences between the unbounded and bounded containers. The original unbounded containers have pragma Preelaborate whereas the new bounded containers have pragma Pure.

The bounded containers for hashed maps and hashed sets are treated somewhat differently to those for the corresponding unbounded containers regarding hashing.

In the case of unbounded containers, the hashing function to be used is left to the user and is provided as an actual generic parameter. For example, in the case of hashed sets, the package specification begins

```
generic
  type Element_Type is private;
  with function Hash(Element: Element_Type)
                                    return Hash_Type;
  with function Equivalent_Elements(Left, Right:
                          Element_Type) return Boolean;
  with function "=" (Left, Right: Element_Type)
                                  return Boolean is <>;
package Ada.Containers.Hashed_Sets is
  pragma Preelaborate(Hashed_Sets);
```

What the implementation actually does with the hash function is entirely up to the implementation. The value returned is in the range of Hash_Type which is a modular type declared in the root package Ada.Containers. The implementation will typically then map this value onto the current range of the capacity in some way. If the unbounded container becomes nearly full then the capacity will be automatically extended and a new mapping will be required; this in turn is likely to require the existing contents to be rehashed. None of this is visible to the user.

In the case of the new bounded containers, these problems do not arise since the capacity is fixed. Moreover, the modulus to be used for the mapping is given when the container is declared since the type has discriminants thus

```
type Set(Capacity: Count_Type;
         Modulus: Hash_Type) is tagged private;
```

The user can then choose the modulus explicitly or alternatively can use the additional function Default_Modulus whose specification is

```
function Default_Modulus(Capacity: Count_Type)
                                    return Hash_Type;
```

This returns an implementation defined value for the number of distinct hash values to be used for the given capacity. Thus we can write

```
My_Set: Set(Capacity => My_Cap;
            Modulus => Default_Modulus(My_Cap));
```

Moreover, for these bounded hashed maps and sets, the function Copy has an extra parameter thus

```
function Copy(Source: Set;
              Capacity: Count_Type := 0;
              Modulus: Hash_Type := 0) return Set;
```

If the capacity is given as zero then the newly returned set has the same capacity as the length of Source as mentioned above. If the modulus is given as zero then the value to be used is obtained by applying Default_Modulus to the new capacity.

As mentioned in the paper on the Predefined Library, Ada 2012 introduces additional functions for hashing strings (fixed, bounded and unbounded) to provide for case insensitive, wide and wide wide situations.

Finally, note that there are no bounded containers for indefinite types. This is because the size of an object of an indefinite type (such as String) is generally not known and so indefinite types need some dynamic storage management. However, the whole point of introducing bounded containers was to avoid such management.

3 Iterating and updating containers

This topic was largely covered in the paper on Iterators and Pools which introduced the generic package Ada.Iterator.Interfaces whose specification is

```
generic
   type Cursor;
   with function Has_Element(Position: Cursor)
                                          return Boolean;
package Ada.Iterator_Interfaces is
   pragma Pure(Iterator_Interfaces);

   type Forward_Iterator is limited interface;
   function First(Object: Forward_Iterator)
                                 return Cursor is abstract;
   function Next(Object: Forward_Iterator;
                 Position: Cursor) return Cursor is abstract;

   type Reversible_Iterator is limited interface and
                                          Forward_Iterator;
   function Last(Object: Reversible_Iterator)
                                 return Cursor is abstract;
   function Previous(Object: Reversible_Iterator;
                 Position: Cursor) return Cursor is abstract;

end Ada.Iterator_Interfaces;
```

This generic package is used by both existing and new container packages. For illustration we consider the list container Ada.Containers.Doubly_Linked_Lists. Here is its specification giving all new and changed material in full (marked -- *12*) and identifying most existing entities by comment only.

```
with Ada.Iterator_Interfaces;                          -- 12
generic
   type Element_Type is private;
   with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Doubly_Linked_Lists is
   pragma Preelaborate(Doubly_Linked_Lists);
   pragma Remote_Types(Doubly_Linked_Lists)        -- 12

   type List is tagged private                         -- 12
      with Constant_Indexing => Constant_Reference,
           Variable_Indexing => Reference,
           Default_Iterator => Iterate,
           Iterator_Element => Element_Type;
   pragma Preelaborable_Initialization(List);
   type Cursor is private;
   pragma Preelaborable_Initialization(Cursor);
   Empty_List: constant List;
   No_Element: constant Cursor;

   function Has_Element(Position: Cursor)    -- moved 12
                                          return Boolean;
   package List_Iterator_Interfaces is                 -- 12
      new Ada.Iterator_Interfaces(Cursor, Has_Element);

   ...   -- functions "=", Length, Is_Empty, Clear, Element
   ...   -- procedures Replace_, Query_, Update_Element

   type Constant_Reference_Type                        -- 12
     (Element: not null access constant Element_Type)
                                          is private
     with Implicit_Dereference => Element;

   type Reference_Type                                 -- 12
     (Element: not null access Element_Type) is private
     with Implicit_Dereference => Element;

   function Constant_Reference                         -- 12
                    (Container: aliased in List;
                     Position: in Cursor)
                  return Constant_Reference_Type;
   function Reference                                  -- 12
                    (Container: aliased in out List;
                     Position: in Cursor)
                          return Reference_Type;

   procedure Assign(Target: in out List;               -- 12
                    Source: in List);

   function Copy(Source: List) return List;            -- 12

   ...   -- Move, Insert, Prepend, Append,
   ...   -- Delete, Delete_First, Delete_Last,
   ...   -- Reverse_Elements, Swap, Swap_Links, Splice,
   ...   -- First, First_Element, Last, Last_Element,
   ...   -- Next, Previous, Find, Reverse_Find,
   ...   -- Contains, Iterate, Reverse_Iterate

   function Iterate(Container: in List) return         -- 12
         List_Iterator_Interfaces.Reversible_Iterator'Class;

   function Iterate(Container: in List;                -- 12
                    Start: in Cursor) return
         List_Iterator_Interfaces.Reversible_Iterator'Class;

   ...   -- generic package Generic_Sorting

private
   ... -- not specified by the language
end Ada.Containers.Doubly_Linked_Lists;
```

Note that the function Has_Element has been moved. In Ada 2005 it was declared towards the end between Contains and Iterate. It has been moved so that it can be used as an actual parameter in the declaration of List_Iterator_Interfaces using the instantiation of Ada.Iterator_Interfaces.

It will be recalled from the paper on Iterators and Pools that in Ada 2012 we can simply write

```
for C in The_List.Iterate loop
  ...                          -- do something via cursor C
end loop;
```

or even

```
for E of The_List loop
  ...                          -- do something to Element E
end loop;
```

rather than the laborious and error prone

```
C: The_List.Cursor;
E: Twin;
F: Forward_Iterator'Class := The_List.Iterate;
...
C := F.First;
loop
  exit when not The_List.Has_Element(C);
  E := The_List.Element(C);
  ...              -- do something to E
  C := F.Next(C);
end loop;
```

Note that in the case of

```
for C in The_List.Iterate loop
  ...                          -- do something via cursor C
end loop;
```

we are not permitted to assign to C since that would upset the mechanism of the loop. There is an analogy with the traditional loop statement. If we write

```
for K in A'Range loop
  A(K) := 0;
end loop;
```

then the language prevents us from making a direct assignment to the loop parameter K.

If we write

```
for E of The_List loop
  ...                          -- do something to Element E
end loop;
```

then we can change the element E unless The_List has been declared as constant.

It will be recalled that subprograms Replace_Element, Query_Element and Update_Element are defined for all containers in Ada 2005. Query_Element and Update_Element permit *in situ* operations. Thus in order to find the value of some component Q of an element of The_List identified by cursor C we can write either

```
X := Element(C).Q;
```

or we can first declare a slave procedure

```
procedure Get_Q(E: in Element_Type) is
begin
  X := E.Q;
end Get_Q;
```

and then call Query_Element thus

```
Query_Element(C, Get_Q'Access);
```

The advantage of the former is that it is easy but it could be slow because it copies the whole element which could be enormous. The advantage of the latter is that it does not copy the element; its disadvantage is that it is somewhat incomprehensible.

In Ada 2012, we can do much better. The type List now has new functions Reference and Constant_Reference, so we can write for example

```
X := The_List.Constant_Reference(C).Q;
```

This works because the function Constant_Reference returns a value of Constant_Reference_Type and this moreover has aspect Implicit_Dereference whose value is Element.

However, we can simplify this even more because the type List has aspects Constant_Indexing and Variable_Indexing which refer to the functions Constant_Reference and Reference. The result is that we can simply write

```
X := The_List(C).Q;                -- gosh that's better
```

which is a lot better than calling Query_Element.

Similarly, if we just want to update the component Q of some element given by a cursor C, then in Ada 2005 we either have to create a whole new element with the new value for Q and then use Replace_Element thus

```
Temp: E_Type := Element(C);
...
Temp.Q := X;

Replace_Element(The_List, C, Temp);
```

or declare a slave procedure and use Update_Element thus

```
procedure Put_Q(E: in out Element_Type) is
begin
  E.Q := X;
end Put_Q;

Update_Element(The_List, C, Put_Q'Access);
```

Again the first is slow, the second is gruesome (well, they are both gruesome really).

In Ada 2012 we simply write

```
The_List(C).Q := X;                -- gosh again
```

which implicitly uses the aspect Variable_Indexing to call the function Reference which gives access to the element.

It will be remembered that there are dire warnings in Ada 2005 about tampering with elements and cursors. Thus we must not use Update_Element (that is via Put_Q in the example above) to do other things such as add new elements.

Although tampering is still possible in Ada 2012; the new features discourage it. Thus if we write

```
The_List(C).Q := X;
```

rather than calling Update_Element then no tampering can occur (unless X is some gruesome function).

Similarly if we write

```
for C in My_Container loop
   ...
   Delete(My_Container, Position => C);    --illegal
   ...
end loop;
```

then we are prevented from madness since the parameter Position of Delete is of mode **in out** and this is not matched by the loop parameter C which is a constant. However, if we write the loop out using First and Next as illustrated earlier then we could get into trouble.

4 Multiway tree containers

Three new containers are added for multiway trees; two correspond to the existing unbounded definite and unbounded indefinite forms for existing structures such as Lists and Maps in Ada 2005. There is also a bounded form corresponding to the newly introduced bounded containers for the existing structures discussed above. As expected their names are

```
A.C.Multiway_Trees
A.C.Indefinite_Multiway_Trees
A.C.Bounded_Multiway_Trees
```

These containers have all the operations required to operate on a tree structure where each node can have multiple child nodes to any depth. Thus there are operations on subtrees, the ability to find siblings, to insert and remove children and so on. It will be noted that many of the operations on trees are similar to corresponding operations on lists.

We will look in detail at the unbounded definite form by giving its specification interspersed with some explanation. It starts with the usual generic parameters.

```
with Ada.Iterator_Interfaces;
generic
   type Element_Type is private;
   with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Multiway_Trees is
   pragma Preelaborate(Multiway_Trees);
   pragma Remote_Types(Multiway_Trees);

   type Tree is tagged private
      with Constant_Indexing => Constant_Reference,
           Variable_Indexing => Reference,
           Default_Iterator => Iterate,
           Iterator_Element => Element_Type;
   pragma Preelaborable_Initialization(Tree);
   type Cursor is private;
   pragma Preelaborable_Initialization(Cursor);
   Empty_Tree: constant Tree;
   No_Element: constant Cursor;

   function Has_Element(Position: Cursor)
                                          return Boolean;

   package Tree_Iterator_Interfaces is new
         Ada.Iterator_Interfaces(Cursor, Has_Element);
```

This is much as expected and follows the same pattern as the start of the list container in the previous section.

```
   function Equal_Subtree(Left_Position: Cursor;
                          Right_Position: Cursor) return Boolean;
   function "=" (Left, Right: Tree) return Boolean;

   function Is_Empty(Container: Tree) return Boolean;

   function Node_Count(Container: Tree)
                                          return Count_Type;
   function Subtree_Node_Count(Position: Cursor)
                                          return Count_Type;

   function Depth(Position: Cursor) return Count_Type;

   function Is_Root(Position: Cursor) return Boolean;
   function Is_Leaf(Position: Cursor) return Boolean;
   function Root(Container: Tree) return Cursor;
   procedure Clear(Container: in out Tree);
```

A tree consists of a set of nodes linked together in a hierarchical manner. Nodes are identified as usual by the value of a cursor. Nodes can have one or more child nodes; the children are ordered so that there is a first child and a last child. Nodes with the same parent are siblings. One node is the root of the tree. If a node has no children then it is a leaf node.

All nodes other than the root node have an associated element whose type is Element_Type. The whole purpose of the tree is of course to give access to these element values in a structured manner.

The function "=" compares two trees and returns true if and only if they have the same structure of nodes and corresponding nodes have the same values as determined by the generic parameter "=" for comparing elements. Similarly, the function Equal_Subtree compares two subtrees.

The function Node_Count gives the number of nodes in a tree. All trees have at least one node, the root node. The function Is_Empty returns true only if the tree consists of just this root node. Note that A_Tree = Empty_Tree, Node_Count(A_Tree) = 1 and Is_Empty(A_Tree) always have the same value. The function Subtree_Node_Count returns the number of nodes in the subtree identified by the cursor. If the cursor value is No_Element then the result is zero.

The functions Is_Root and Is_Leaf indicate whether a node is the root or a leaf respectively. If a tree is empty and so consists of just a root node then that node is both the root and a leaf so both functions return true.

The function Depth returns 1 if the node is the root, and otherwise indicates the number of ancestor nodes. Thus a node which is an immediate child of the root has depth equal to 2. The function Root returns the cursor designating the root of a tree. The procedure Clear removes all elements from the tree so that it consists just of a root node.

```
function Element(Position: Cursor)
                                    return Element_Type;

procedure Replace_Element(Container: in out Tree;
                          Position: in Cursor;
                          New_Item: in Element_Type);

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

procedure Update_Element(Container: in out Tree;
                         Position: in Cursor;
                         Process: not null access procedure
                           (Element: in out Element_Type));
```

These subprograms have the expected behaviour similar to other containers.

```
type Constant Reference_Type
  (Element: not null access constant Element_Type)
                                              is private
  with Implicit_Dereference => Element;

type Reference_Type
  (Element: not null access Element_Type) is private
  with Implicit_Dereference => Element;

function Constant_Reference
                        (Container: aliased in Tree;
                         Position: in Cursor)
                      return Constant_Reference_Type;

function Reference(Container: aliased in out Tree;
                   Position: in Cursor)
                                 return Reference_Type;
```

These types and functions are similar to those for the other containers and were explained in the paper on Iterators and Pools and also in the previous section.

```
procedure Assign(Target: in out Tree;
                 Source: in Tree);

function Copy(Source: Tree) return Tree;

procedure Move(Target: in out Tree;
               Source: in out Tree);
```

The subprograms Assign and Copy behave as expected and were explained in the section on Bounded and Unbounded containers. The procedure Move moves all the nodes from the source to the target after first clearing the target; it does not make copies of the elements so after the operation the source only has a root node.

```
procedure Delete_Leaf(Container: in out Tree;
                      Position: in out Cursor);

procedure Delete_Subtree(Container: in out Tree;
                         Position: in out Cursor);

procedure Swap(Container: in out Tree;
                         I, J: in Cursor);
```

The procedures Delete_Leaf and Delete_Subtree check that the cursor value designates a node of the container and raise Program_Error if it does not. Program_Error is also raised if Position designates the root node and so cannot be removed. In the case of Delete_Leaf, if the node has any children then Constraint_Error is raised. The appropriate nodes are then deleted and Position is set to No_Element.

The procedure Swap interchanges the values in the two elements denoted by the two cursors. The elements must be in the given container (and must not denote the root) otherwise Program_Error is raised.

```
function Find(Container: Tree; Item: Element_Type)
                                    return Cursor;

function Find_In_Subtree(Item: Element_Type;
                         Position: Cursor)
                                    return Cursor;

function Ancestor_Find(Item: Element_Type;
                       Position: Cursor)
                                    return Cursor;

function Contains(Container: Tree;
                  Item: Element_Type) return Boolean;
```

These search for an element in the container with the given value Item. The function Contains returns false if the item is not found; the other functions return No_Element if the item is not found. The function Find searches the whole tree starting at the root node, Find_In_Subtree searches the subtree rooted at the node given by Position including the node itself; these searches are in depth-first order. The function Ancestor_Find searches upwards through the ancestors of the node given by Position including the node itself.

Depth-first order is explained at the end of the section.

```
procedure Iterate(Container: in Tree;
                  Process: not null access procedure
                    (Position: in Cursor));

procedure Iterate_Subtree(Position: in Cursor;
                  Process: not null access procedure
                    (Position: in Cursor));
```

These apply the procedure designated by the parameter Process to each element of the whole tree or the subtree. This includes the node at the subtree but not at the root; iteration is in depth-first order.

```
function Iterate(Container: in Tree) return
      Tree_Iterator_Interfaces.Forward_Iterator'Class;

function Iterate_Subtree(Position: in Cursor) return
      Tree_Iterator_Interfaces.Forward_Iterator'Class;
```

The first of these is called if we write

```
for C in The_Tree.Iterate loop
  ...                        -- do something via cursor C
end loop;
```

and iterates over the whole tree in the usual depth-first order. In order to iterate over a subtree we write

```
for C in The_Tree.Iterate(S) loop
  ...                        -- do something via cursor C
end loop;
```

and this iterates over the subtree rooted at the cursor position given by S.

If we use the other new form of loop using **of** thus

```
for E of The_Tree loop
   ...                          -- do something to element E
end loop;
```

then this also calls Iterate since the aspect Default_Iterator of the type Tree (see above) is Iterate. However, we cannot iterate over a subtree using this mechanism.

```
function Child_Count(Parent: Cursor)
                                    return Count_Type;

function Child_Depth(Parent, Child: Cursor)
                                    return Count_Type;
```

The function Child_Count returns the number of child nodes of the node denoted by Parent. This count covers immediate children only and not grandchildren.

The function Child_Depth indicates how many ancestors there are from Child to Parent. If Child is an immediate child of Parent then the result is 1; if it is a grandchild then 2 and so on.

```
procedure Insert_Child(Container: in out Tree;
                       Parent: in Cursor;
                       Before: in Cursor;
                       New_Item: in Element_Type;
                       Count: in Count_Type := 1);

procedure Insert_Child(Container: in out Tree;
                       Parent: in Cursor;
                       Before: in Cursor;
                       New_Item: in Element_Type;
                       Position: out Cursor;
                       Count: in Count_Type := 1);

procedure Insert_Child(Container: in out Tree;
                       Parent: in Cursor;
                       Before: in Cursor;
                       Position: out Cursor;
                       Count: in Count_Type := 1);
```

These three procedures enable one or more new child nodes to be inserted. The parent node is given by Parent. If Parent already has children then the new nodes are inserted before the child node identified by Before; if Before is No_Element then the new nodes are inserted after all existing children. The second procedure is similar to the first but also returns a cursor to the first of the added nodes. The third is like the second but the new elements take their default values. Note the default value of one for the number of new nodes.

```
procedure Prepend_Child(Container: in out Tree;
                        Parent: in Cursor;
                        New_Item: in Element_Type;
                        Count: in Count_Type := 1);

procedure Append_Child(Container: in out Tree;
                       Parent: in Cursor;
                       New_Item: in Element_Type;
                       Count: in Count_Type:= 1);
```

These insert the new children before or after any existing children.

```
procedure Delete_Children(Container: in out Tree;
                          Parent: in Cursor);
```

This procedure simply deletes all the children, grandchildren, and so on of the node designated by Parent.

```
procedure Copy_Subtree(Target: in out Tree;
                       Parent: in Cursor;
                       Before: in Cursor;
                       Source: in Cursor);
```

This copies the complete subtree rooted at Source into the tree denoted by Tree as a subtree of Parent at the place denoted by Before using the same rules as Insert_Child. Note that this makes a complete copy and creates new nodes with values equal to the corresponding existing nodes. Note also that Source might be within Tree but might not. There are the usual various checks.

```
procedure Splice_Subtree(Target: in out Tree;
                         Parent: in Cursor;
                         Before: in Cursor;
                         Source: in out Tree;
                         Position: in out Cursor);

procedure Splice_Subtree(Container: in out Tree;
                         Parent: in Cursor;
                         Before: in Cursor;
                         Position: in Cursor);

procedure Splice_Children(Target: in out Tree;
                          Target_Parent: in Cursor;
                          Before: in Cursor;
                          Source: in out Tree;
                          Source_Parent: in Cursor);

procedure Splice_Children(Container: in out Tree;
                          Target_Parent: in Cursor;
                          Before: in Cursor;
                          Source_Parent: in Cursor);
```

These are similar to the procedures Splice applying to lists. They enable nodes to be moved without copying. The destination is indicated by Parent or Target_Parent together with Before as usual indicating where the moved nodes are to be placed with respect to existing children of Parent or Target_Parent.

The first Splice_Subtree moves the subtree rooted at Position in the tree Source to be a child of Parent in the tree Target. Note that Position is updated to be the appropriate element of Target. We can use this procedure to move a subtree within a tree but an attempt to create circularities raises Program_Error.

The second Slice_Subtree is similar but only moves a subtree within a container. Again, circularities cannot be created.

The procedures Splice_Children are similar but move all the children and their descendants of Source_Parent to be children of Target_Parent.

```
function Parent(Position: Cursor) return Cursor;
function First_Child(Parent: Cursor) return Cursor;
function First_Child_Element(Parent: Cursor)
                                    return Element_Type;
function Last_Child(Parent: Cursor) return Cursor;
function Last_Child_Element(Parent: Cursor)
                                    return Element_Type;
function Next_Sibling(Position: Cursor) return Cursor;
function Previous_Sibling(Position: Cursor)
                                    return Cursor;
procedure Next_Sibling(Position: in out Cursor);
procedure Previous_Sibling(Position: in out Cursor);
```

Hopefully, the purpose of these is self-evident.

```
procedure Iterate_Children(Parent: in Cursor;
                  Process: not null access procedure
                                    (Position: in Cursor));

procedure Reverse_Iterate_Children
                                    (Parent : in Cursor;
                  Process: not null access procedure
                                    (Position: in Cursor));
```

These apply the procedure designated by the parameter Process to each child of the node given by Parent. The procedure Iterate_Children starts with the first child and ends with the last child whereas Reverse_Iterate_Children starts with the last child and ends with the first child. Note that these do not iterate over grandchildren.

```
function Iterate_Children(Container: in Tree;
                                Parent: in Cursor) return
    Tree_Iterator_Interfaces.Reversible_Iterator'Class;
```

This is called if we write

```
for C in Parent.Iterate_Children loop
   ...                          -- do something via cursor C
end loop;
```

and iterates over all the children from Parent.First_Child to Parent.Last_Child. Note that we could also insert **reverse** thus

```
for C in reverse Parent.Iterate_Children loop
   ...                          -- do something via cursor C
end loop;
```

in which case the iteration goes in reverse from Parent.Last_Child to Parent.First_Child. The observant reader will note that this function returns Reversible_Iterator'Class and so can go in either direction whereas the functions Iterate and Iterate_Subtree described earlier use Forward_Iterator'Class and cannot be reversed.

```
private
   ...              -- not specified by the language
end Ada.Containers.Multiway_Trees;
```

The above descriptions have not described all the situations in which something can go wrong and so raise Constraint_Error or Program_Error. Generally, the former is raised if a source or target is No_Element; the latter is raised if a cursor does not belong to the appropriate tree. In particular, as mentioned above, an attempt to create an illegal tree such as one with circularities using Splice_Subtree raises Program_Error. Remember also that every tree has a root node but the root node has no element value; attempts to remove the root node or read its value or assign a value similarly raise Program_Error.

The containers for indefinite and bounded trees are much as expected.

In the case of the indefinite tree container the generic formal type is

```
type Element_Type(<>) is private;
```

The other significant difference is that the procedure Insert_Child without the parameter New_Item is omitted; this is because indefinite types do not have a default value.

In the case of the bounded tree container the changes are similar to those for the other containers. One change is that the package has pragma Pure; the other changes concern the capacity. The type Tree is

```
type Tree(Capacity: Count_Type) is tagged private;
```

and the function Copy is

```
function Copy(Source: Tree;
                    Capacity: Count_Type := 0) return Tree;
```

And of course the exception Capacity_Error is raised in various circumstances.

Applications of trees are usually fairly complex. The tree structure for depicting the analysis of a program for a whole language such as even Ada 83 has an enormous variety of nodes corresponding to the various syntactic structures. And trees depicting human relationships are complex because of multiple marriages, divorces, illegitimacy and so on. So we content ourselves with a couple of small examples.

A tree representing a simple algebraic expression involving just the binary operations of addition, subtraction, multiplication and division applied to simple variables and real literals is straightforward. Nodes are of three kinds, those representing operations have two children giving the two operands, and those representing variables and literals have no children and so are leaf nodes.

We can declare the element type thus

```
type Operator is ('+', '–', '×', '/');
type Kind is (Op, Var, Lit);

type El(K: Kind) is
  record
    case K is
      when Op =>
        Fn: Operator;
      when Var =>
        V: Character;
      when Lit =>
```

```
      Val: Float;
   end case;
end record;
```

Note that the variables are (as typically in mathematics) represented by single letters. So the expression

$(x + 3) \times (y - 4)$

is represented by nodes with elements such as

```
(Op, '×')
(Var, 'x')
(Lit, 3.0)
```

So now we can declare a suitable tree thus

```
package Expression_Trees is
   new Ada.Containers.Multiway_Trees(El);

use Expression_Trees;

My_Tree: Tree := Empty_Tree;

C: Cursor;
```

and then build it by the following statements

```
C := Root(My_Tree);

Insert_Child(Container => My_Tree,
             Parent => C,
             Before => No_Element,
             New_Item => (Op, '×'),
             Position => C);
```

This puts in the first real node as a child of the root which is designated by the cursor C. There are no existing children so Before is No_Element. The New_Item is as mentioned earlier. Finally, the cursor C is changed to designate the position of the newly inserted node.

We can then insert the two children of this node which represent the mathematical operations + and –.

```
Insert_Child(My_Tree, C, No_Element, (Op, '+'));
Insert_Child(My_Tree, C, No_Element, (Op, '–'));
```

These calls are to a different overloading of Insert_Child and have not changed the cursor. The second call also has Before equal to No_Element and so the second child goes after the first child. We now change the cursor to that of the first newly inserted child and then insert its children which represent *x* and 3. Thus

```
C := First_Child(C);
Insert_Child(My_Tree, C, No_Element, (Var, 'x'));
Insert_Child(My_Tree, C, No_Element, (Lit, 3.0));
```

And then we can complete the tree by inserting the final two nodes thus

```
C := Next_Sibling(C);
Insert_Child(My_Tree, C, No_Element, (Var, 'y'));
Insert_Child(My_Tree, C, No_Element, (Lit, 4.0));
```

Of course a compiler will do all this recursively and keep track of the cursor rather more neatly than we have in this manual illustration.

The resulting tree should be as in Figure 1.

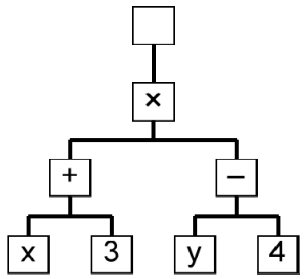


Figure 1 The expression tree

We can assume that the variables are held in an array which might be as follows

```
subtype Variable_Name is Character range 'a' .. 'z';

Variables: array (Variable_Name) of Float;
```

We can then evaluate the tree by a recursive function such as

```
function Eval(C: Cursor) return Float is
   E: El := Element(C);
   L, R: Float
begin
   case E.K is
      when Op =>
         L := Eval(First_Child(C));
         R := Eval(Last_Child(C));
         case E.Fn is
            when '+' => return (L+R);
            when '–' => return (L–R);
            when '×' => return (L*R);
            when '/' => return (L/R);
         end case;
      when Var =>
         return Variables(E.V);
      when Lit =>
         return E.Val;
   end case;
end Eval;
```

Finally, we obtain the value of the tree by

```
X := Eval(First_Child(Root(My_Tree)));
```

Remember that the node at the root has no element so hence the call of First_Child.

An alternative approach would be to use tagged types with a different type for each kind of node rather than the variant record. This would be much more flexible and would have required the use of the unbounded indefinite container Ada.Containers.Indefinite_Multiway_Trees.

As a more human example we can consider the family tree of the Tudor Kings and Queens of England. We start with Henry VII, who had four children, Arthur, Margaret, Henry VIII and Mary. See Figure 2.

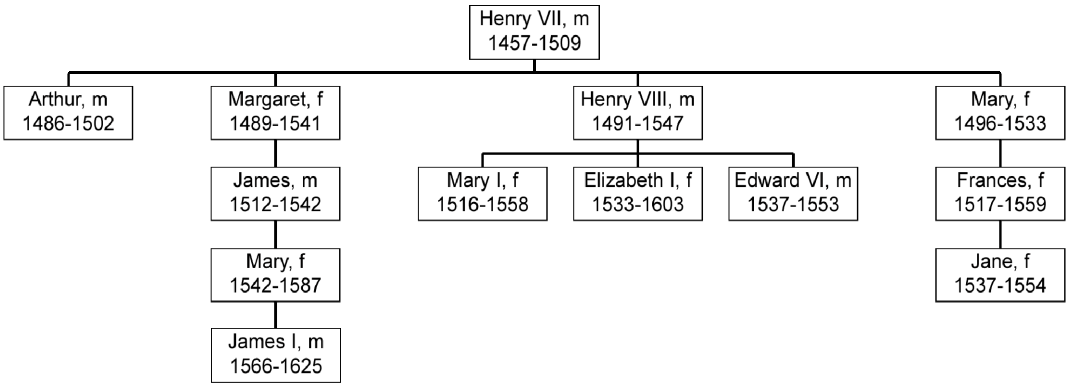


Figure 2 The Tudor tree

Arthur died young, Margaret married James IV of Scotland and had James (who was thus James V of Scotland), Henry VIII had three children, namely Edward VI, Mary I and Elizabeth I. And Mary had Frances. Henry VII was succeeded by Henry VIII and he was succeeded by his three children.

Remember the rules of primogeniture. The heir is the eldest son if there are sons; if not then the heir is the eldest daughter. If there are no offspring at all then we go back a generation and try again. Hence Edward VI became king despite being younger than Mary.

Since Edward, Mary and Elizabeth had no children we go back to the descendants of the other children of Henry VII. Margaret, her son James, and his daughter Mary Queen of Scots were all dead by then, so the throne of England went to the son of Mary who became James I of England and VI of Scotland and thus united the two thrones. So the Tudor line died with Elizabeth (Good Queen Bess).

Incidentally, Frances, the daughter of Mary, the fourth child of Henry VII, had a daughter, Lady Jane Grey; she was Queen for 9 days but lost her head over a row with Mary I.

Representing this is tricky, especially with people such as Henry VIII having so many wives. But the essence could be represented by a tree with a simple element type thus

```
type Person is
  record
    Name: String(1 .. 10);
    Sex: Gender;
    Birth: Date;
    Death: Date;
  end record;
```

With such a structure and the dates, starting from Henry VII and using the rules of primogeniture, one should be able to trace the monarchs (apart from poor Lady Jane Grey who would I am sure much rather not have been involved).

The overall tree structure is shown in Figure 2.

With the obvious connections we can define useful functions such as

```
function Are_Cousins(A, B: Cursor) return Boolean is
  (Parent(A) /= Parent (B) and then
                Parent(Parent(A)) = Parent(Parent(B)));
```

More of a challenge is to define a function Is_Successor using the rules described above. The reader can contemplate these and other family relationships and attempt to construct the Tudor tree.

Finally, an explanation of depth-first order. The general principle is that child nodes are visited in order before their parent. We can symbolically write this as

```
procedure Do_Node(N) is
begin
  for CN in N.First_Child .. N.Last_Child loop
    Do_Node(CN);
  end loop;
  if not N.Is_Root then
    Do_Element(N);
  end if;
end Do_Node;
```

and the whole thing is triggered by calling Do_Node(Root). Remember that the root node has no element. The result is that the first element to be processed is that of the leftmost leaf.

Thus in the tree illustrated below in Figure 3, the elements are visited in order A, B, C, D, and so on. Note that the root has no element and so is not visited.

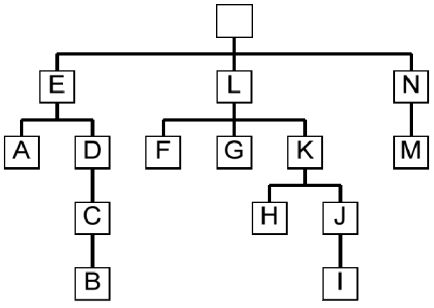


Figure 3 A tree showing depth-order first

5 The holder container

As mentioned in the Introduction, it is not possible to declare an object of an indefinite type that can hold any value of that type since the object becomes constrained by the mandatory initial value. Thus we can write

```
Pet: String := "dog";
```

We can assign "cat" to Pet but we cannot assign "rabbit" because it is too long.

This is overcome in Ada 2012 by the introduction of the holder container which can hold a single indefinite object. Its specification is

```
generic
   type Element_Type(<>) is private;
   with function "=" (Left, Right: Element_Type)
                                    return Boolean is <>;
package Ada.Containers.Indefinite_Holders is
   pragma Preelaborate(Indefinite_Holders);
   pragma Remote_Types(Indefinite_Holders);

   type Holder is tagged private;
   pragma Preelaborable_Initialization(Holder);

   Empty_Holder: constant Holder;

   function "=" (Left, Right: Holder) return Boolean;

   function To_Holder(New_Item: Element_Type)
                                          return Holder;

   function Is_Empty(Container: Holder) return Boolean;

   procedure Clear(Container: in out Holder);

   function Element(Container: Holder)
                                    return Element_Type;

   procedure Replace_Element(Container: in out Holder;
                             New_Item: in Element_Type);

   procedure Query_Element(Container: in Holder;
                           Process: not null access procedure
                             (Element: in Element_Type));

   procedure Update_Element(Container: in out Holder;
                            Process: not null access procedure
                              (Element: in out Element_Type));

   type Constant_Reference_Type
     (Element: not null access constant Element_Type)
                                                 is private
     with Implicit_Dereference => Element;

   type Reference_Type
     (Element: not null access Element_Type) is private
     with Implicit_Dereference => Element;

   function Constant_Reference
                             (Container: aliased in Holder)
                           return Constant_Reference_Type;

   function Reference(Container: aliased in out Holder)
                                    return Reference_Type;

   procedure Assign(Target: in out Holder;
                    Source: in Holder);

   function Copy(Source: Holder) return Holder;

   procedure Move(Target: in out Holder;
                  Source: in out Holder);

private
   ...              -- not specified by the language
end Ada.Containers.Indefinite_Holders;
```

Hopefully, the purpose of the facilities provided by this container are obvious given an understanding of the use of the existing containers. It would be possible to use a list container with just a single element to act as a holder but it seems better to have an explicit container with probably less overhead and risk of confusion.

A trivial example of its use might be to provide a holder for pets. We write

```
package Strings is
      new Ada.Containers.Indefinite_Holders(String);

Kennel: Strings.Holder := To_Holder("cat");
```

This declares an object Kennel which is a wrapper for a string and initializes it with the string "cat". We can replace the cat with a rabbit by writing

```
Kennel := To_Holder("rabbit");
```

However, using To_Holder in this way could be a bit slow since this will create a new object which has to be destroyed after the assignment. It is better to write

```
Replace_Element(Kennel, "rabbit");
```

If we want to print out the contents of the kennel we just write

```
Put(Element(Kennel));
```

Operations such as Update_Element are provided partly for uniformity but also because the object might be large so that it is better to update it *in situ*. Alternatively, we can use the functions such as Reference as explained earlier.

6 Queue containers

When the goals of the revision to Ada 2005 were discussed, one of the expectations was that it would be possible to improve the containers, or maybe introduce variants, that would be task safe. However, further investigation revealed that this would not be practicable because the number of ways in which several tasks could interact with a container such as a list or map was large.

However, one data structure that is amenable to controlled access by several tasks is the queue. One or more tasks can place objects on a queue and one or more can remove them. Moreover, the existing container library did not include queues as such so we were not tied to any existing structures.

There are in fact four different queue containers in Ada 2012. These are all for elements of a definite type. Two are bounded and two are unbounded. And there are priority and synchronized queues. The names are

A.C.Unbounded_Synchronized_Queues
A.C.Bounded_Synchronized_Queues
A.C.Unbounded_Priority_Queues
A.C.Bounded_Priority_Queues

At one stage it was also planned to have unbounded containers for elements of an indefinite type. This would then have been similar to the other containers which have unbounded definite, unbounded indefinite and bounded definite forms. However, there were significant problems with the Dequeue operation to remove an indefinite object related to the fact that Ada does not have entry functions. This is easily overcome by making the elements of the queue a holder container as described in the previous section.

These four different queue containers are all derived from a single synchronized interface declared in a generic package whose specification is as follows

```
generic
   type Element_Type is private;              -- definite
package A.C.Synchronized_Queue_Interfaces is
   pragma Pure(Synchronized_Queue_Interfaces);

   type Queue is synchronized interface;

   procedure Enqueue(Container: in out Queue;
                     New_Item: in Element_Type)
                                              is abstract
      with Synchronization => By_Entry;

   procedure Dequeue(Container: in out Queue;
                     Element: out Element_Type)
                                              is abstract
      with Synchronization => By_Entry;

   function Current_Use(Container: Queue)
                     return Count_Type is abstract;
   function Peak_Use(Container: Queue)
                     return Count_Type is abstract;
end A.C.Synchronized_Queue_Interfaces;
```

This generic package declares the synchronized interface Queue and four operations on queues. These are the procedures Enqueue and Dequeue to add items to a queue and remove items from a queue respectively; note the aspect Synchronization which ensures that all implementations of these abstract procedures must be by an entry. There are also functions Current_Use and Peak_Use which can be used to monitor the number of items on a queue.

The four queue containers are generic packages which themselves declare a type Queue derived in turn from the interface Queue declared in the package above. We will look first at the synchronized queues and then at the priority queues.

The package for unbounded synchronized queues is as follows

```
with System; use System;
with A.C.Synchronized_Queue_Interfaces;
generic
   with package Queue_Interfaces is new
                  A.C.Synchronized_Queue_Interfaces(<>);
   Default_Ceiling: Any_Priority := Priority'Last;
package A.C.Unbounded_Synchronized_Queues is
   pragma Preelaborate
                  (Unbounded_Synchronized_Queues);

   package Implementation is
      ...                    -- not specified by the language
   end Implementation;

   protected type Queue(Ceiling: Any_Priority :=
                                    Default_Ceiling)
                  with Priority => Ceiling
                    is new Queue_Interfaces.Queue with
      overriding
      entry Enqueue(New_Item:
                     in Queue_Interfaces.Element_Type);
      overriding
      entry Dequeue(Element:
                    out Queue_Interfaces.Element_Type);

      overriding
      function Current_Use return Count_Type;
      overriding
      function Peak_Use return Count_Type;

   private
      ...                    -- not specified by the language
   end Queue;

private
   ...              -- not specified by the language
end A.C.Unbounded_Synchronized_Queues;
```

Note that there are two generic parameters. The first (Queue_Interfaces) has to be an instantiation of the interface generic Synchronized_Queue_Interfaces; remember that the parameter (<>) means that any instantiation will do. The second parameter concerns priority and has a default value so we can ignore it for the moment.

Inside this package there is a protected type Queue which controls access to the queues via its entries Enqueue and Dequeue. This protected type is derived from Queue_Interfaces.Queue and so promises to implement the operations Enqueue, Dequeue, Current_Use and Peak_Use of that interface. And indeed it does implement them and moreover implements Enqueue and Dequeue by entries as required by the aspect Synchronization.

As an example suppose we wish to create a queue of some records such as

```
type Rec is record ... end record;
```

First of all we instantiate the interface package (using named notation for clarity) thus

```
package Rec_Interface is
   new A.C.Synchronized_Queue_Interfaces
                                 (Element_Type => Rec);
```

This creates an interface from which we can create various queuing mechanisms for dealing with objects of the type Rec.

Thus we might write

```
package Unbounded_Rec_Package is
  new A.C.Unbounded_Synchronized_Queues
                    (Queue_Interfaces => Rec_Interface);
```

Finally, we can declare a protected object, My_Rec_UQ which is the actual queue, thus

```
My_Rec_UQ: Unbounded_Rec_Package.Queue;
```

To place an object on the queue we can write

```
Enqueue(My_Rec_UQ, Some_Rec);
```

or perhaps more neatly

```
My_Rec_UQ.Enqueue(Some_Rec);
```

And to remove an item from the queue we can write

```
My_Rec_UQ.Dequeue(The_Rec);
```

where The_Rec is some object of type Rec which thereby is given the value removed.

The statement

```
N := Current_Use(My_Rec_UQ);
```

assigns to N the number of items on the queue when Current_Use was called (it could be out of date by the time it gets into N) and similarly Peak_Use(My_Rec_UQ) gives the maximum number of items that have been on the queue since it was declared.

This is all task safe because of the protected type; several tasks can place items on the queue and several, perhaps the same, can remove items from the queue without interference.

It should also be noticed that since the queue is unbounded, we never get blocked by Enqueue since extra storage is allocated as required just as for the other unbounded containers (I suppose we might get Storage_Error).

The observant reader will note the mysterious local package called Implementation. This enables the implementation to declare local types to be used by the protected type. It will be recalled that there is an old rule that one cannot declare a type within a type. These local types really ought to be within the private part of the protected type; maybe this is something for Ada 2020.

The package for bounded synchronized queues is very similar. The only differences (apart from its name) are that it has an additional generic parameter Default_Capacity and the protected type Queue has an additional discriminant Capacity. So its specification is

```
with System; use System;
with A.C.Synchronized_Queue_Interfaces;
generic
  with package Queue_Interfaces is new
                A.C.Synchronized_Queue_Interfaces(<>);
  Default_Capacity: Count_Type;
  Default_Ceiling: Any_Priority := Priority'Last;
package A.C.Bounded_Synchronized_Queues is
  pragma Preelaborate
                (Bounded_Synchronized_Queues);

  package Implementation is
    ...                    -- not specified by the language
  end Implementation;

  protected type Queue(Capacity: Count_Type :=
                                    Default_Capacity,
                       Ceiling: Any_Priority :=
                                    Default_Ceiling)
          with Priority => Ceiling
            is new Queue_Interfaces.Queue with

  ...                      -- etc as for the unbounded one

end A.C.Bounded_Synchronized_Queues;
```

So using the same example, we can use the same interface package Rec_Interface. Now suppose we wish to declare a bounded queue with capacity 1000, we can write

```
package Bounded_Rec_Package is
  new A.C.Bounded_Synchronized_Queues
                (Queue_Interfaces => Rec_Interface
                 Default_Capacity => 1000);
```

Finally, we can declare a protected object, My_Rec_BQ which is the actual queue, thus

```
My_Rec_BQ: Bounded_Rec_Package.Queue;
```

And then we can use the queue as before. To place an object on the queue we can write

```
My_Rec_BQ.Enqueue(Some_Rec);
```

And to remove an item from the queue we can write

```
My_Rec_BQ.Dequeue(The_Rec);
```

The major difference is that if the queue becomes full then calling Enqueue will block the calling task until some other task calls Dequeue. Thus, unlike the other containers, Capacity_Error is never raised.

Note that having given a value for Default_Capacity, it can be overridden when the queue is declared, perhaps

```
My_Rec_Giant_BQ:
   Bounded_Rec_Package.Queue(Capacity => 100000);
```

These packages also provide control over the ceiling priority of the protected type. By default it is Priority'Last. This default can be overridden by our own default when the queue package is instantiated and can be further specified as a discriminant when the actual queue object is declared. So we might write

```
My_Rec_Ceiling_BQ:
   Bounded_Rec_Package.Queue(Ceiling => 10);
```

In the case of the bounded queue, if we do not give an explicit capacity then the ceiling has to be given using named notation. This does not apply to the unbounded

queue which only has one discriminant, so to give that a ceiling priority we can just write

```
My_Rec_Ceiling_UQ:
   Unbounded_Rec_Package.Queue(10);
```

But clearly the use of the named notation is advisable.

Being able to give default discriminants is very convenient. In Ada 2005, this was not possible if the type was tagged. However, in Ada 2012, it is permitted in the case of limited tagged types and a protected type is considered to be limited. This was explained in detail in the paper on Structure and Visibility.

If we wanted to make a queue of indefinite objects, then as mentioned above, there is no special container for this because Dequeue would be difficult to use since it is a procedure and not a function. So the actual parameter would have to be constrained which means knowing before the call the value of the discriminant, tag, or bound of the object which is unlikely. However, we can use the holder container to wrap the indefinite type so that it looks definite.

So to create a queue for strings, using the example of the previous section, we can write

```
package Strings is
      new Ada.Containers.Indefinite_Holders(String);

package Strings_Interface is
  new A.C.Synchronized_Queue_Interfaces
            (Element_Type => Strings.Holder);

package Unbounded_Strings_Package is
  new A.C.Unbounded_Synchronized_Queues
             (Queue_Interfaces => Strings_Interface);
```

and then finally declare the actual queue

```
My_Strings_UQ: Unbounded_Strings_Package.Queue;
```

To put some strings on this queue, we write

```
My_Strings_UQ.Enqueue(To_Holder("rabbit"));

My_Strings_UQ.Enqueue(To_Holder("horse"));
```

or even

```
My_Strings_UQ.Enqueue(Element(Kennel));
```

We now turn to considering the two other forms of queue which are the unbounded and bounded priority queues.

Here is the specification of the unbounded priority queue

```
with System; use System;
with A.C.Synchronized_Queue_Interfaces;
generic
  with package Queue_Interfaces is new
            A.C.Synchronized_Queue_Interfaces(<>);

  type Queue_Priority is private;
  with function Get_Priority
   (Element : Queue_Interfaces.Element_Type)
                        return Queue_Priority is <>;
  with function Before(Left, Right : Queue_Priority)
                        return Boolean is <>;

  Default_Ceiling: Any_Priority := Priority'Last;
package A.C.Unbounded_Priority_Queues is
  pragma Preelaborate(Unbounded_Priority_Queues);

  package Implementation is
    ...                    -- not specified by the language
  end Implementation;

  protected type Queue(Ceiling: Any_Priority :=
                                   Default_Ceiling)
          with Priority => Ceiling
            is new Queue_Interfaces.Queue with

    overriding
    entry Enqueue(New_Item:
                  in Queue_Interfaces.Element_Type);
    overriding
    entry Dequeue(Element:
                 out Queue_Interfaces.Element_Type);

    not overriding
    procedure Dequeue_Only_High_Priority
      (At_Least: in Queue_Priority;
       Element: in out Queue_Interfaces.Element_Type;
       Success: out Boolean);

    overriding
    function Current_Use return Count_Type;
    overriding
    function Peak_Use return Count_Type;

  private
    ...                    -- not specified by the language
  end Queue;

private
  ...              -- not specified by the language
end A.C.Unbounded_Priority_Queues;
```

The differences from the synchronized bounded queue are that there are several additional generic parameters, namely the private type Queue_Priority and the two functions Get_Priority and Before which operate on objects of the type Queue_Priority, and also that the protected type Queue has an additional operation, the protected procedure Dequeue_Only_High_Priority.

The general idea is that elements have an associated priority which can be ascertained by calling the function Get_Priority. The meaning of this priority is given by the function Before.

When we call Enqueue, the new item is placed in the queue taking due account of its priority with respect to other elements already on the queue. So it will go before all less important elements as defined by Before. If existing elements already have the same priority then it goes after them.

As expected Dequeue just returns the first item on the queue and will block if the queue is empty.

The new procedure Dequeue_Only_High_Priority (note that it is marked as **not overriding** unlike the other operations) is designed to enable us to process items only if they are important enough as defined by the parameter At_Least. The priority of the first element E on the queue is P as given by Get_Priority(E). And so if Before(At_Least, P) is false, then the item on the queue is indeed important enough and so is removed from the queue and the Boolean parameter Success is set to true. On the other hand if Before(At_Least, P) is true then the item is not removed and Success is set to false. Note especially that Dequeue_Only_High_Priority never blocks. If the queue is empty, then Success is just set to false; it never waits for an item to be put on the queue.

As an (unrealistic) example, suppose we decide to make the queue of strings into a priority queue and that the priority is given by their length so that "rabbit" takes precedence over "horse". Remember that the type of the elements is Strings.Holder. We can define the priority as given by the attribute Length so we might as well make the actual type corresponding to Queue_Priority as simply Natural. Then we define

```
function S_Get_Priority(H: Strings.Holder)
                                        return Natural is
  (H.Element'Length);

function S_Before(L, R: Natural) return Boolean is
  (L > R);
```

Note the convenient use of expression functions for this sort of thing.

The instantiation now becomes

```
package Unbounded_Priority_Strings_Package is
  new A.C.Unbounded_Priority_Queues
              (Queue_Interfaces => Strings_Interface,
               Queue_Priority => Natural,
               Get_Priority => S_Get_Priority,
               Before => S_Before);
```

and we then declare a queue thus

```
My_Strings_UPQ:
        Unbounded_Priority_Strings_Package.Queue;
```

To put some strings on this queue, we write

```
My_Strings_UPQ.Enqueue(To_Holder("rabbit"));

My_Strings_UPQ.Enqueue(To_Holder("horse"));

My_Strings_UPQ.Enqueue(To_Holder("donkey"));

My_Strings_UPQ.Enqueue(To_Holder("gorilla"));
```

The result is that "gorilla" will have jumped to the head of the queue despite having been put on last. It will be followed by "rabbit" and "donkey" and the "horse" is last.

If we do

```
My_Strings_UPQ.Dequeue_Only_High_Priority
                                  (7, Kennel, OK);
```

then the "gorilla" will be taken from the queue and placed in the Kennel and OK will be true. But if we then do it again, nothing will happen because the resulting head of the queue (the "rabbit") is not long enough.

Finally, we need to consider bounded priority queues. They are exactly like the unbounded priority queues except that they have the same additional features regarding capacity as found in the synchronized queues. Thus the only differences (apart from the name) are that there is an additional generic parameter Default_Capacity and the protected type Queue has an additional discriminant Capacity.

As a final example we will do a bounded priority queue of records. Suppose the records concern requests for servicing a dishwasher. They might included usual information such as the model number, name and address of owner and so on. They might also have a component indicating degree of urgency, such as

Urgent – machine has vomited dirty water all over floor; housewife/husband having a tantrum,

Major – machine won't do anything; husband refuses to help with washing up,

Minor – machine leaves some dishes unclean, mother-in-law is coming next week,

Routine – machine needs annual service.

So we might have

```
type Degree is (Urgent, Major, Minor, Routine);

type Dish_Job is
  record
    Urgency: Degree;
    Name: ...
    ...
  end record;
```

First we declare the interface for this type

```
package Dish_Interface is
  new A.C.Synchronized_Queue_Interfaces
                          (Element_Type => Dish_Job);
```

and then we declare the two slave functions for the priority mechanism thus

```
function W_Get_Priority(X: Dish_Job) return Degree is
  (X.Urgency);

function W_Before(L, R: Degree) return Boolean is
  (Degree'Pos(L) < Degree'Pos(R));
```

The instantiation is then

```
package Washer_Package is
  new A.C.Bounded_Priority_Queues
            (Queue_Interfaces => Dish_Interface,
             Queue_Priority => Degree,
             Get_Priority => W_Get_Priority,
             Before => W_Before,
             Default_Capacity => 100);
```

and we declare the queue of waiting calls thus

```
Dish_Queue: Washer_Package.Queue;
```

which gives a queue with the default capacity of 100.

The staff taking requests then place the calls on the queue by

```
Dish_Queue.Enqueue(New_Job);
```

To cope with the possibility that the queue is full, they can do a timed entry call; remember that this is possible because the procedure Enqueue in the interface package has Synchronization => By_Entry.

And then general operatives checking in and taking the next job do

```
Dish_Queue.Dequeue(Next_Job);
```

However, at weekends we can suppose that just one operative is on call and deals with only Urgent and Major calls. He might check the queue from time to time by calling

```
Dish_Queue.Dequeue_Only_High_Priority
                                (Major, My_Job, Got_Job);
```

and if Got_Job is false, he can relax and go back to digging the garden or playing golf.

7 Sorting

Ada 2005 provides two containers for sorting arrays; one is for unconstrained array types and one is for constrained array types. The specification of the unconstrained one is

```
generic
  type Index_Type is (<>);
  type Element_Type is private;
  type Array_Type is
        array (Index_Type range <>) of Element_Type;
  with function "<" (Left, Right: Element_Type)
                                    return Boolean is <>;
procedure Ada.Containers.Generic_Array_Sort
                          (Container: in out Array_Type);
pragma Pure(Ada.Containers.Generic_Array_Sort);
```

This does the obvious thing. It sorts the array Container so that the components are in the order defined by the generic parameter "<".

We could for example sort the letters in a string into alphabetical order. We would declare

```
procedure String_Sort is
  new Ada.Containers.Generic_Array_Sort
                          (Positive, Character, String);
```

and then if we had a string such as

```
Bigpet: String := "rabbit";
```

we could apply String_Sort to it thus

```
String_Sort(Bigpet);
```

and the value in Bigpet will now be "abbirt".

That is all in Ada 2005. However, sorting doesn't just apply to arrays and Ada 2012 provides a much more flexible approach. An additional container is provided whose specification is

```
generic
  type Index_Type is (<>);
  with function Before(Left, Right: Index_Type)
                                    return Boolean;
  with procedure Swap(Left, Right: in Index_Type);
procedure Ada.Containers.Generic_Sort
                          (First, Last: Index_Type'Base);
pragma Pure(Ada.Containers.Generic_Sort);
```

This can be used to sort any indexable structure and not just arrays. The generic parameters define the required ordering through the parameter Before much as expected. The cunning trick however, is that the means of interchanging two items in the structure is provided by the parameter Swap.

As an illustration we can use this on the array Bigpet. We can use an expression function for BP_Before and so we write

```
function BP_Before(L, R: Positive) return Boolean is
  (Bigpet(L) < Bigpet(R));

procedure BP_Swap(L, R: in Positive) is
  Temp: Character;
begin
  Temp := Bigpet(L);
  Bigpet(L) := Bigpet(R);
  Bigpet(R) := Temp;
end BP_Swap;

procedure BP_Sort is
  new Ada.Containers.Generic_Sort
                          (Positive, BP_Before, BP_Swap);
```

and then we actually do the sort by

```
BP_Sort(Bigpet'First, Bigpet'Last);
```

That may seem to be rather a struggle but the key point is that the technique can be used to sort items in any indexable structure such as a vector container.

Suppose we have a number of records of a type Score which might be

```
type Score is
  record
    N: Natural := 0;
    OS: Other_Stuff;
  end record;
```

and we declare a vector container to hold such objects thus

```
package Scores is
  new Ada.Containers.Vectors(Natural, Score);

My_Vector: Scores.Vector;
```

Now assume that we have added various objects of the type Score to our vector and that we decide that we would like them sorted into order determined by their component N.

We write

```
function MV_Before(L, R: Natural) return Boolean is
  (Scores.Element(My_Vector, L).N <
                    Scores.Element(My_Vector, R).N);

procedure MV_Swap(L, R: in Natural) is
begin
  Scores.Swap(My_Vector, L, R);
end MV_Swap;

procedure MV_Sort is
              new Ada.Containers.Generic_Sort
                    (Natural, MV_Before, MV_Swap);
```

and then we do the sort by

```
MV_Sort(Scores.First_Index(My_Vector),
                      Scores.Last_Index(My_Vector));
```

Note that the vectors container package conveniently already has a procedure Swap.

This vector example is not very exciting because it might be recalled that the vectors containers already have their own internal generic sort. To use it on this example we would have to write

```
package MV_Sorting is
  new Scores.Generic_Sorting(MV_Before);

MV_Sorting.Sort(My_Vector);
```

which is somewhat simpler. However, note that this sorts the whole vector. If we only wanted to sort part of it, say from elements in index range P to Q then it cannot be used. But that would be easy with the new one since we would simply write

```
MV_Sort(P, Q);
```

Note that curiously this does not need to mention My_Vector.

8 Streaming

Ada 2005 was somewhat unclear regarding streaming values from and to containers. This is clarified in Ada 2012. Thus if V is a vector container then V'Write writes Length(V) elements to the stream concerned.

An important point is that in order to simplify the interchange between containers, we are assured that we can stream between them using 'Write and 'Read. Thus we can stream between a bounded and an unbounded container as well as between two bounded or two unbounded containers provided of course that the elements all have the same subtype.

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*.

Overview of the 15th International Real-Time Ada Workshop

14-16 September 2011
Liébana (Cantabria), Spain

Contents *

Workshop Session Summaries

- J. Real, J.F. Ruiz, “Session Summary: Multiprocessor Issues, part 1”
- A. Wellings, L.M. Pinho, “Session Summary: Multiprocessor Issues, part 2 resource control protocols)”
- A. Burns, T. Vardanega, “Session Summary: Language Profile and Application Frameworks"
- J.A. de la Puente, S. Michell, “Session Summary: Concurrency Issues”

Program Committee

Mario Aldea Rivas (Program Chair), Neil Audsley, John Barnes, Ben Brosgol, Alan Burns, Michael González Harbour (Local Chair), José Javier Gutiérrez, Stephen Michell, Brad Moore, Luís Miguel Pinho, Juan Antonio de la Puente, Jorge Real, Jose F. Ruiz, Joyce Tokar, Tullio Vardanega, Andy Wellings and Rod White.

Workshop Participants

Mario Aldea Rivas, University of Cantabria, Spain
António Barros, Polytechnic Institute of Porto, Portugal
Alan Burns, University of York, UK
Michael González Harbour, University of Cantabria, Spain
Javier Gutierrez, University of Cantabria, Spain
Stephen Michell, Maurya Software, Canada
Marco Panunzio, University of Padua, Italy
Hector Pérez Tijero, University of Cantabria, Spain
Luis Miguel Pinho, Polytechnic Institute of Porto, Portugal
Juan Antonio de la Puente, Technical University of Madrid, Spain
Jorge Real, Universitat Politècnica de València, Spain
José Ruiz, AdaCore, France
Sergio Saez, Universitat Politècnica de València, Spain
Lin Shiyao, University of York, UK
Joyce Tokar, Pyrrhus Software, USA
Tullio Vardanega, University of Padua, Italy
Andy Wellings, University of York, UK
Rod White, MBDA, UK
Juan Zamorano, Technical University of Madrid, Spain

Sponsors

* The Proceedings of the 15th International Real-Time Ada Workshop are published in ACM Ada Letters, Volume XXXII, Number 1, April 2013.

Session Summary: Multiprocessor Issues, part 1*

Chair: Jorge Real

Rapporteur: José F. Ruiz

1 Introduction

The topic of multiprocessors was addressed by quite a number of position papers this year, so the whole first day of the IRTAW 15 workshop, while we were all fresh, was allocated to discussing multiprocessor issues. There was a discussion about dispatching domains and multiprocessor architectures during the morning, focusing on resource control protocols in the afternoon.

This summary addresses the morning session, whose goals were to review and evaluate the Ada 2012 support for multiprocessors, and think about possible additions to future (post 2012) language revisions. Specific issues discussed in this session were:

- The current definition of dispatching domains
- Per dispatching domain scheduling policies
- Dynamic dispatching domains
- Support for very large number of cores
- Non-SMP architectures
- Deferred attributes

The following sections will provide the details of the discussions around these subjects.

2 Dispatching domains in Ada 2012

Ada 2012 support for task affinities is provided by dispatching domains (Ada 2012 RM D.16.1), and there are already reference implementations [6] indicating that the current definition is appropriate and implementable on top of operating systems and kernels.

These early implementations showed an editorial error in the current definition of package System.Multiprocessors.Dispatching_Domains, which is intended to be Preelaborate, but it cannot be so because it depends on package Ada.Real_Time which is not Preelaborate. Alan Burns volunteered to submit this minor correction to ARG.

The definition of the System_Dispatching_Domain was found slightly misleading, because it is defined as constant (although it is implicitly modified by the creation of other dispatching domains), and it is not always a contiguous range as the other dispatching domains. Hence, for the System_Dispatching_Domain, Get_First_CPU and Get_Last_CPU do not make much sense, and Assign_Task cannot be used. However, these minor annoyances could probably be addressed better in the Rationale than by changes in the Reference Manual.

3 Dispatching domain's scheduling policy

The position paper submitted by Alan Burns and Andy Wellings [2] to last IRTAW 14 advocated for a more flexible definition of dispatching domains, where one could assign specific scheduling policies to each dispatching domains.

The discussion around this feature showed that this behavior can be achieved by carefully using the existing support for dispatching domains; if the priority range is partitioned into non-overlapping priority bands, each of these bands to be used only by tasks allocated to the same dispatching domain, we can set specific dispatching policies for the different ranges of priorities (and hence for each dispatching domain).

Therefore, there was not a strong motivation for trying to push forward this feature.

4 Dynamic dispatching domains

Ada 2012 defines dispatching domains as static entities. Therefore, mode changes involving migration of CPUs from one dispatching domain to another cannot be implemented. Reconfiguring dispatching domains when the underlying hardware changes (CPUs being added or removed to/from an Ada partition) is not supported either.

It would be interesting to support dynamic dispatching domains to address these situations. Note that the current Ada 2012 model assumes that the set of processors available to an Ada partition remains constant for its whole lifetime, so addressing changes to the hardware is outside the intention in the language. Still, the existing static support is not flexible enough for some kinds of mode changes which would be desirable.

The position papers discussed in the workshop did not address this functionality, so after discussing for a while its implications it was decided to encourage the submission of concrete proposals for the next workshop, motivating the need and detailing its design.

5 Very large number of cores

Luís Miguel Pinho started this discussion by stating that Ada is not fit for a large number of cores and we should start thinking about how to address this limitation. Tasks

* The Proceedings of the 15th International Real-Time Ada Workshop are published in ACM Ada Letters, Volume XXXII, Number 1, April 2013.

and protected objects are too heavyweight, in terms of time to create and destroy, context switch, synchronization protocol, etc.

When there are many CPUs available, what would be desirable is to have the notion of "parallel" activity, which could be functions (without side effects), blocks, loops, or others.

This notion of user-level _fine-grained parallelism is already present in many programming languages and libraries. In ParaSail [8], language semantics are parallel by default, and subprograms, loops, statements and some other elements can be executed in parallel. Intel's Cilk [1] extends the C language with primitives to express parallelism, and the Cilk run-time system maps the expressed parallelism into a parallel execution framework. Cilk provides keywords to indicate how subprograms can be executed in parallel. Hence, there exist today different languages, libraries and APIs to express parallelism in a program. The field of automatic parallelization by compilers or tools has not yet been proved as very efficient, leaving user-level description of parallelism as the only means to achieve highly parallel applications effectively.

Therefore, the workshop identified this topic as a relevant one. Ada has been prominent in treating concurrent units (tasks) as first-class citizens in the language, and it would be good to address better support for parallelism. The submission of proposals about this subject is strongly encouraged for the next workshop.

6 Non-SMP architectures

Ada 2012 support for multiprocessor architectures focuses on Symmetric MultiProcessor (SMP) architectures, where all processors are identical and access to main memory is uniform (caching aside). In the Ada model a partition is a single address space.

At the last IRTAW 14 workshop there was a position paper by Andy Wellings et al. [9] addressing the difficulties to handle this kind of hardware architectures. The main issues explained there:

- Understanding the address map. The first requirement is to be able to represent the platform. The goal is to include the notion of locality and distance (in number of hops) of a processor from a memory bank servicing a particular address, so that tasks can allocate the objects they use in a "close" location. It would require to add the concepts of memory bank and distance to Ada. Looking at standard ways to describe memory topologies, it was suggested to look at Portable Hardware Locality (hwloc) [4], which provides a portable abstraction of the hierarchical hardware topology. It would be useful to have an interface to a library to get the hardware representation.
- Using storage pools to partition the heap. The idea is to allow users to allocate objects at addresses where they can be accessed more efficiently, for example by choosing a local memory bank. Storage pools appear as the obviuos choice to do it, but the issue is that you cannot set the address of a storage pool; Ada lets you partition the heap into separate storage pools, but you cannot specify their address location. The workshop agreed it would be interesting to be able to specify the address attribute of a storage pool. POSIX has typed memory facilities [5] to control the allocation of memory pools.

Hence, it was considered appealing to add support to represent memory topologies, and to be able to use this information to allocate objects in an informed manner.

We considered as well whether it would be good to have the means to indicate that an object is only used by tasks executing on the same processor. This information could be used to indicate that all those tasks share the same cache, and hence enforcement of cache coherence would not be needed. However, we realized that this is not an issue because the hardware would enforce cache coherence only when needed, and if two tasks are in the same CPU, data can remain in cache without being written to memory (it would be written to memory if another core tries to read/write this memory area).

7 Deferred attributes

During this session, Sergio Sáez stated the existing limitations of the current model of setting attributes (priority, deadline and affinity). Possible ways to address this issue where discussed in the session about "Concurrency Issues" [3].

Priority, deadline and affinity can be changed dynamically (Set_{Priority,Deadline,CPU}) and all of these are dispatching points. Ada 2012 allows you to change deadline and affinity after a delay (Delay_Until_And_Set_{Deadline,CPU}), but it is not possible to set more than one of these attributes atomically. When these task attributes are not changed atomically, some scheduling artifacts could arise giving rise to incorrect schedules [7]. We agreed that we want to be able to atomically change task's attributes to avoid unintended effects.

Possible solutions using timing events or protected objects to perform these changes atomically were discussed. However, they could not meet the requirements when changing another task's attributes.

The addition of subprograms for setting attributes at the next dispatching point (such as Set_Next_{Priority, Deadline,CPU}) was proposed and discussed in more detail during the "Concurrency Issues" session.

8 Conclusion

There was a very interesting and constructive discussion during all the morning session (continued during the afternoon and the rest of the workshop) about the challenges raised by the new multiprocessor architectures.

The first important outcome to point out is that the current support for multiprocessors in Ada 2012 (dispatching domains) is considered very appropriate. In addition to that, it matches rather smoothly the support typically offered by multiprocessor operating systems.

Two mechanisms were explored to make dispatching domains more flexible. One was the possibility of setting dispatching domain's scheduling policies, which was deemed not very important since it can be achieved with the current support. The second was to migrate CPUs among dispatching domains, and was considered interesting for implementing mode changes. This workshop encouraged the submission of proposals addressing these dynamic dispatching domains.

Another subject considered very appealing for future workshops was the support for user-level fine grained parallelism. As more and more CPUs are made available, the notion of task is perhaps not the right abstraction, with respect to controlling parallelism with smaller granularity.

It was also agreed that Ada provides limited expressive power for non-SMP architectures, and it would be good to add more control over the physical address map.

Finally, better support for deferred attributes was motivated during this morning session, but lunch time deferred the discussion of the required interface until a later session . . .

References

[1] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou, (1995), *Cilk: An efficient multithreaded runtime system*, Journal of Parallel and Distributed Computing, pages 207-216.

[2] Alan Burns and Andy J. Wellings (2010), *Supporting execution on multiprocessor platforms*, Ada Letters, XXX(1):16-25.

[3] Juan A. de la Puente and Stephen Michell (2011), *Session summary: Concurrency issues*, This issue.

[4] Portable hardware locality (hwloc). Available at http://www.open-mpi.org/projects/hwloc.

[5] IEEE. *Portable Operating System Interface* (POSIX) - *Part 1: System Application Program Interface (API) - Advanced Realtime Extensions [C Language],* 2000. 1003.1j-2000.

[6] José F. Ruiz (2011), *Going real-time with Ada 2012 and GNAT*, Ada Letters.

[7] Sergio Sáez and Alfons Crespo (2011), *Deferred setting of scheduling attributes in ada 2012*, Ada Letters.

[8] Tucker Taft (2011), *Multicore programming in ParaSail*, In A. Romanovsky and T. Vardanega, editors, Ada-Europe 2011, 16th International Conference on Reliable Software Technologies, volume 6652 of Lecture Notes in Computer Science (LNCS). Springer.

[9] Andy Wellings, Abdul H Malik, Neil Audsley, and Alan Burns (2010), *Ada and cc-NUMA architectures: What can be achieved with Ada 2005?,* Ada Letters, XXX(1):125-134.

Session Summary: Multiprocessor Issues, part 2 (resource control protocols)*

Chair: Andy Wellings

Rapporteur: Luís Miguel Pinho

1 Introduction

The second session on the topic of Multiprocessor Issues took place after a post-lunch relaxing walk on *Picos de Europa*, 800 meters above the workshop location and more than 1800 meters above the sea level. Indeed a stimulating activity to foster a productive session.

The goals of this session were:

A. To review and evaluate the efficacy of the Ada 2012 support in the area of multiprocessor resource control
   - The first topic being to analyze if the Reference Manual (RM) for Ada 2012 had been fully updated concerning priority inheritance issues, given that Ada 2012 allows tasks to be fully partitioned, globally scheduled or scheduled in disjoint domains;
   - A second topic was the wording of priority inheritance for the case of Earliest Deadline Scheduling in multiprocessors, where a potential issue had been put forward by [1].
   - The third topic was to evaluate whether Protected Objects should be used only for local resource access or if new access protocols were needed to allow it to be used globally without the need for spin-locks, following a position paper on the topic [1];

B. To look beyond Protected Objects and Rendezvous to other paradigms amenable to be used in multiprocessor platforms, for instance Software Transactional Memory (STM) or wait-free queues
   - The objective was to analyze to what extent the Ada primitives are suitable to implement these new paradigms and if secondary standards were the appropriate mechanism to introduce these paradigms into Ada;
   - In particular, STM would be analyzed in more detail as there was a position paper on the topic [2].

C. To review previous workshops proposals of new synchronization primitives to improve parallel execution of Ada programs.
   - Following a proposal from [3] the workshop decided to revisit broadcast primitives for calling PO subprograms in parallel [4, 5] and the use of parallel barriers in Protected Objects [4, 6].

2 Ada 2012 support in the area of multiprocessor resource control

2.1 Review of priority inheritance and task partitioning

In order to provide a basis for discussion, the Session Chair started by presenting an overview of the specification of priority inheritance (Ada 2012 RM D.1):

- *During activation, a task being activated inherits the active priority that its activator (see 9.2) had at the time the activation was initiated.*
- *During rendezvous, the task accepting the entry call inherits the priority of the entry call (see 9.5.3 and D.4).*
- *During a protected action on a protected object, a task inherits the ceiling priority of the protected object (see 9.5 and D.3).*

The chair then introduced into the discussion if the meaning of priority inheritance would still hold in a fully partitioned system (when the task inheriting the priority is on a different processor to the task which is waiting) or in a cluster-based system (when the task inheriting the priority is in a different dispatching domain to the task which is waiting). Would inheritance of priorities among domains be meaningful, if tasks were interacting in two different dispatching domains, using different priority ranges?

A note was nevertheless made that the current Ada model, albeit allowing multiple dispatching domains, imposes a single scheduling policy in all domains. Therefore, after several rounds of discussion, there was an agreement that priority assignments in all processors should be globally coherent. If that approach is followed, inheritance of a priority from one processor to the other is always correct.

*The Proceedings of the 15th International Real-Time Ada Workshop are published in ACM Ada Letters, Volume XXXII, Number 1, April 2013.

2.2 EDF, priority inheritance and Multiprocessors

In this short topic, [1] noted a potential misleading wording of paragraph 26 in section D.2.6, that could allow programs to behave incorrectly:

> *For a task T to which policy EDF_Across_Priorities applies, the base priority is not a source of priority inheritance; the active priority when first activated or while it is blocked is defined as the maximum of the following:*
>
> *[...]*
>
> - *the highest priority P, if any, less than the base priority of T such that one or more tasks are executing within a protected object with ceiling priority P and task T has an earlier deadline than all such tasks; and furthermore T has an earlier deadline than all other tasks on ready queues with priorities in the given EDF_Across_Priorities range that are strictly less than P.*

The workshop discussed the issue, and although there was not an agreement that the sentence was incorrect, it was agreed that Alan Burns would further analyze the issue after the workshop to check if the wording would need to be clarified.

2.3 Access protocols for Protected Objects in multiprocessors

In the third topic of the session, the Chair started by providing an overview of the Reference Manual wordings concerning the access and control protocols for Protected Objects, noting that both the RM and the Annotated Reference Manual (ARM) do not fully define the access protocol for a protected object on a multiprocessor system.

For instance, Note 19, Section 9.5.1 states:

> *If two tasks both try to start a protected action on a protected object, and at most one is calling a protected function, then only one of the tasks can proceed. Although the other task cannot proceed, it is not considered blocked, and it might be consuming processing resources while it awaits its turn. There is no language-defined ordering or queuing presumed for tasks competing to start a protected action on a multiprocessor such tasks might use busy-waiting; for monoprocessor considerations, see D.3, "Priority Ceiling Locking".*
>
> *Discussion: The intended implementation on a multi-processor is in terms of "spin locks" – the waiting task will spin."*

While in D.2.1, paragraph 3, it states:

> *It is implementation defined whether, on a multiprocessor, a task that is waiting for access to a protected object keeps its processor busy.*

Thus it is implementation defined whether spinning occurs non-preemptively or, if not, at what priority it is performed. Furthermore, it is not defined whether there are queues (FIFO or priority) associated with the spin-lock. The workshop agreed that this could pose a potential problem to analyze maximum blocking times.

Afterwards, Andy Wellings performed a review of the most current used shared data protocols for multiprocessor systems, analyzing if these could be implemented in Ada. From the perform analysis, the conclusion was the majority of these protocols could not be used, as they were based in suspending contending tasks in FIFO or Priority Queues, something that was disallowed in Ada.

At this point Michael Gonzalez noted that there was nothing in Ada preventing an implementation based on suspending on the lock. Andy noted that the RM states that access control in multiprocessors should be done with spin-locks. After a brief discussion, the general view was that although the RM promoted the spin-lock model, it allowed implementations to provide alternative implementations.

Alan Burns then proposed that if there was a recognized good solution for access control protocols based in suspension in the lock, the RM wording could be changed in the next revision process.

The session then went to analyze if it would be useful to allow the programmer the ability to change the underlying locking code (thus controlling the implementation). That would allow trying different protocols or to use the best fit for a particular applications.

Both Alan Burns and Tullio Vardanega noted that the current model in Ada is the result of many years of research which provided a sound access model for the monoprocessor case. The current work for the multiprocessor case was still far from that and there was no clear winner at the moment. The workshop came back to the issue of the suitability of the concurrency model of Ada for multiprocessors, but the general feeling was that there was yet no general agreement on what the model would be and what protocols to support.

A note was also made by José Ruiz that usually the access model of the Protected Objects uses mechanisms which are provided by the underlying Operating System. It would not be possible to give the programmer direct access to those. However, Andy noted that the idea being proposed was different, in that the approach would be to provide applications with an interface to specify application protocols that the runtime could used instead of the one provided by the underlying operating system.

Then Andy presented a proposal [1] of an API that allowed to control and extend the queue locks, and implement the access control protocols:

```
package System.Multiprocessors.Queue_Locks is
  type Queue_Order is
       (FIFO_Ordered, Priority_Ordered);
```

```
type Spinning_Priority is (Active_Priority_Of_Task,
                           Non_Preemptively);
type Spin_Lock(
      Length : Positive := 1;
       Order : Queue_Order := FIFO_Ordered;
       At_Pri : Spinning_Priority := Non_Preemptively)
       is private;
function Acquire(L : Spin_Lock) return Boolean
procedure Release(L : Spin_Lock);

type Suspension_Lock(
       Length : Positive := 1;
       Order : Queue_Order := FIFO_Ordered )
      is private;

function Acquire(L : Suspension_Lock)
      return Boolean;
function Remove_Head(L : Suspension_Lock)
     return Task_Id;
procedure Add(L : Suspension_Lock; T : Task_Id);
function Highest_Queued_Priority
     (L : Suspension_Lock) return Any_Priority;
procedure Release(L :  Suspension_Lock);

private
  ...
end System.Multiprocessors.Queue_Locks;

package
System.Multiprocessors.Protected_Object_Access is
 type Lock_Type is (Read, Write);
 type Lock_Visibility is (Local, Global);
 type Protected_Controlled is
       new Limited_Controlled with private;
 overriding procedure Initialize
       (C : in out Protected_Controlled);
 overriding procedure Finalize
      (C : in out Protected_Controlled);
 procedure Lock
      (C : in out Protected_Controlled; L : Lock_Type;
       V : Lock_Visibility;  Ceiling : Priority;
       Tid :  Task_Id);
 procedure Unlock (C : in out Protected_Controlled;
                   Tid :  Task_Id);
private
 ...
end
System.Multiprocessors.Protected_Object_Access;
```

Simultaneously an Aspect in the Protected Objects would allow the programmer to specify that access protocol was user defined:

```
protected type X (PC : access Protected_Controlled)
      with Locking_Policy => (User_Protected,
                              Lock_Visibility => Global,
                              Locking_Algorithm => PC) is
 procedure A;
end X;
```

Several issues were still open, such as if the proposed functionality would be sufficient to build the current and future protocols or if spin-locks would still be predefined, but the advantage of this model was allowing to use better algorithms (and test new ones) and still be allowed to use Protected Objects as the data sharing mechanism. Furthermore, compilers can reject or ignore unsupported aspects/pragmas.

To summarize, two approaches could be made available to allow programmers to specify access protocols:

- One would be to use low-level abstractions (such as locks), being Protected Objects not used for multiprocessors;
- The second being Protected Objects augmented with user defined access protocols.

The general agreement of the workshop was that the latter would be the better approach. The workshop view is that this is a worthwhile idea that should be further exploited and presented in the next IRTAW.

3 Looking beyond Protected Objects

3.1 Software Transactional Memory

In the second part of the session, Miguel Pinho started by presenting an overview of Transactional Memory (TM), providing a quick overview of how in this approach atomic sections are executed concurrently and speculatively, in isolation. In particular, Miguel briefly explained how transactions worked with multiple versions of the data objects, allowing increasing the parallelization of execution.

Transactional Memory provides a higher-abstraction to programmers, which can focus on the algorithms, writing the sequential code and identifying the atomic sections. The underlying TM mechanism then controls the concurrent interaction of the transactions.

Studies show that TM is an alternative to locks for larger number of cores, particularly under low contention, when there is a predominance of read-only or short running transactions and there is a low ratio of pre-emptions during its execution. Nevertheless, TM suffers from extra overheads, such as the time needed to access data (it is not accessed directly and updates may be aborted) and the extra memory needed for having multiple versions of the data.

A proposal was being made to support Software TM, as, although less efficient that its hardware counterpart, it was more flexible and more implementations and research was being performed. The proposal was for a standard API which would allow programs to be independent of particular STM implementations and algorithms, and that would also allow to test new contention protocols.

Miguel then presented an example of a potential (conceptual) implementation from [2], currently only addressing non-nested transactions:

```
-- we need a transaction identifier structure
My_Transaction : Transaction;
-- start an update transaction
```

```
My_Transaction.Start(Update);
loop
  -- read a value from a transactional object
  x := trans_object_1.Get(My_Transaction);
  -- write a value to a transactional object
  trans_object_2.Set(My_Transaction, y);
  -- try to commit transaction
  exit when My_Transaction.Commit;
exception
  -- handle possible exceptions here...
end loop;

-- Transactional object
package Transactional_Objects is
  type Transactional_Object is tagged private;
  -- examples of transactional class methods
  procedure Set(T_Object: Transactional_Object;
                Transaction_ID : Transaction;
                Value : Object_Type);
  function Get(T_Object: Transactional_Object;
               Transaction_ID : Transaction)
        return Object_Type;
private
  type Transactional_Object is tagged
  record
    Current_Value : Object_Type;
    Accesses_Set : <list of pointers to transaction
                    identifiers>
    -- some other relevant fields...
  end record;
end Transactional_Objects;

type Transaction_Type is (Read_Only, Update);
-- Transaction identifier
package Transactions is
  type Transaction is tagged private;
  procedure Start(T : Transaction;
                  TRX_Type : Transaction_Type);
  procedure Abort(T : Transaction);
  procedure Terminate(T : Transaction);
  function Commit(T : Transaction)
        return Boolean;
private
  type Transaction is tagged
  record
    Data_Set : List_Ref_Transactional_Objects;
  end record;
end Transactions;
```

In the following discussion, the workshop made a note that the initial read snapshots and the final commit operations would need to be actually performed in memory, so transactional data (only the original not the multi-versions) would need to be volatile.

A doubt was also raised if the implementation of STM would require changes to the language or if standard Ada provides all mechanisms to allow such implementation. It was concluded that the proposed API model is possible to implement, but other models (e.g. declaring transactional objects at the language level) could be interesting to address. It would be also important to incorporate in the proposal the capability for the user to implement its own contention control algorithm.

As a general conclusion, the work was encouraged by the workshop, and it was considered that a prototype implementation would be important to evaluate the proposal. Steve Michell then raised the issue that if a solution was achieved it could be released as a Technical Report or Technical Specification so that initial implementation and evaluations could be performed. Nevertheless, Joyce Tokar noted that it would not be possible if syntactic changes to the language were required – this can only be done through the RM.

4 Mechanisms for improved parallelism

4.1 Broadcast of operations to an array of POs

The first topic in the third part of the session concerned in proposal that was made at IRTAW 13 [4] to support a parallel broadcast of calls to an array of Protected Objects:

```
protected type po_type is
  procedure call(val : integer);
end po_type;

po_array : array (1..10) of po_type;
po_array(7).call(37);          -- single instance call
po_array.all.call(25);         -- broadcast to all
                               -- elements of po array
po_array(1..10).call(25);      -- alternative broadcast to all
                               -- elements of po array
po_array(2..5).call(13);       -- broadcast to restricted
                               -- range of elements
```

One issue that was debated was how the model could allow broadcasts with different data for each actual object. A proposal was also made to consider the use of synchronized interfaces to provide more flexibility to the broadcast.

Afterwards, the usefulness of this mechanism was largely discussed, particularly because the parallel calls would require some execution context (thus it related to the discussion on the previous session on the Ada parallel model). The initial proposal had been performed in the context of direct compilation of Ada code to hardware where these parallel calls could be directly mapped in the FPGA.

The final conclusion was that this was not considered to be currently needed.

4.2 Parallel barriers in functions on POs

The final topic of the session was to revisit the use of parallel barriers in functions within Protected Objects. In the previous workshop the incorporation of parallel barriers had been considerably discussed. The discussion had been separated in both defining a simple task parallel barrier mechanism similar to suspension objects, and allowing tasks in Protected Objects special entries to be

released in parallel. Although the former has made it to Ada 2012 (Synchronous Barriers), the latter, although several rounds of discussion and multiple proposals being made (see [6]), had not reached a conclusion.

A note was made that one of the difficulties with implementing this type of barriers was the data passing issue. POSIX provides simple barriers, because some hardware platforms directly support them. Nevertheless, this does not include data sharing, so this would be difficult to implement.

Finally, the workshop agreed that it would be good to have a model similar to barriers, but more generic and allowing more complex interactions than Synchronous Barriers (and data passing), but that a suitable approach needs further investigation.

5 Conclusions

The session was mainly devoted to analyze how the ubiquitous multicore and multiprocessor platforms impacted the Ada mode for resource control among tasks:

- In the first topic analyzed (priority inheritance and task partitioning) the workshop concluded that the assignment of priorities in partitioned approaches had to be globally coherent and that in this case, priority inheritance between tasks in different domains was always correct.
- In the second topic (priority inheritance under EDF), the workshop felt that there was a possible misleading wording of the behaviour of priority inheritance under EDF, and it was decided to analyzed this further to propose clarification if required.
- In the third topic (access protocols for Protected Objects) the workshop considered important that users are given an interface to control and define different access protocols than simple spin-locks. Further work in this topic is encouraged.
- In the fourth topic (Software Transactional Memory), the workshop concluded that work on other paradigms for concurrency interaction with larger number of cores was important. Further work in this topic is encouraged.
- In the fifth topic (broadcast calls for Protected Object arrays), the workshop considered that currently there was not a need for this mechanism.
- Finally, in the last topic (parallel barriers in Protected Objects), the workshop concluded that this would be a good mechanism to have, but that a suitable approach needs further investigation.

References

[1] S. Lin, A.J. Wellings and A. Burns (2013), *Ada 2012, Resource Sharing and Multiprocessors*, Ada Letters.

[2] A. Barros and L. M. Pinho (2013), *Revisiting Transactions in Ada,* Ada Letters.

[3] A Burns and A. J. Wellings (2013), *Support for Multiprocessor Platforms,* Ada Letters.

[4] M. Ward and N. C. Audsley (2007), *Suggestions for stream based parallel systems in Ada,* Ada Letters, Proceedings of the 13th International Real-Time Ada Workshop, XXVII(2).

[5] J. Real and S. Mitchell (2007), *Beyond Ada 2005 session report.* Ada Letters, Proceedings of the 13th International Real-Time Ada Workshop, XXVII(2).

[6] T. Vardanega, M. González Harbour, L. M. Pinho (2010), *Session Summary: Language and Distribution Issues*, Ada Letters, Proceedings of the 14th International Real-Time Ada Workshop, XXX(1).

Session Summary: Language Profile and Application Frameworks*

Chair: Alan Burns

Rapporteur: Tullio Vardanega

1 Introduction

The Chair's introduction enumerates the issues raised by the papers assigned to the session:

- Beyond Ravenscar – other profiles
- Ravenscar and distribution
- Ravenscar and EDF
- Code archetypes for Ravenscar
- Real-time framework – dealing with multiprocessors and mode changes.

The initial group's perception is that the attention should focus first on the discussion of new language profiles, which is bound to require the largest fraction of the time duration of the session. The group agrees to this proposal, and the Chair presents the highlights of the profile proposal that Burns and Wellings made in [1].

2 Language profiles beyond Ravenscar

An element of the rationale for looking beyond Ravenscar is to avoid the feature creep phenomenon that may diminish the distinctive nature – and the measurable success – of the Ravenscar Profile (RP). The existence of the full language and its wealth of features should be considered to specify new profiles with an expressive power not necessarily close to the end of the RP.

The group understands that there exist two fundamental needs behind language profiles of interest to IRTAW, and to its constituency: (a) to have a coherent set of functionalities; (b) to warrant ease and efficiency of implementation, and, possibly, amenability to certification, although not necessarily to the highest level.

Andy Wellings illustrates the profile proposal made in [1]. The envisioned profile has a twofold motivation:

(1) To gain the ability to tolerate timing faults, which Ravenscar is poorly equipped for, since its fundamental strength is the assurance of absence of them.

(2) To address the greater uncertainty in timing analysis typical of multicore computing.

The direction taken by Burns and Wellings proposal is to incorporate in the profile sufficient means for software dynamic fault tolerance (as per Anderson and Lee's model in [2]): error detection, damage confinement and assessment, error recovery, fault treatment and continued service.

The discussion reviews some variants of fault-error-failure chains, all essentially based on the following, recurrent causal chain:

- Error: WCET overrun or blocking duration overrun (as a ramification of WCET underestimation).
- Error propagation: deadline miss.
- Failure: untimely delivery of service.

The detection means that can be deployed to counter the above chain range from the extreme of deadline miss detection – which however is not a necessary consequence of error propagation, perhaps because of slack capacity available due to less frequent arrival of sporadic tasks in the system – to budget time overrun detection, including blocking time, for better damage confinement, via monitoring the frequency of arrival of sporadic tasks.

The Ravenscar Profile does not allow the use of Ada.Execution_Time.Timers (D.14.1): as a consequence, one can only poll for overrun detection, which obviously is not satisfactory.

Frequency of sporadic arrivals cannot be controlled at language level, but can by the application, for example by forcing task suspension before waiting for the next release event.

Monitoring for overrun of blocking time is no standard provision for either operating system or Ada. The risk is that this fault may go undetected.

For damage confinement one could use budget servers, which the Ravenscar Profile does not support.

The error recovery strategies may include: stopping the overrunning task and then starting a new task to resume service; using the programmatic interface of the asynchronous task control package, D.11. Implementing them under the constraints of the Ravenscar Profile, which does not allow any of those features, calls for

* The Proceedings of the 15th International Real-Time Ada Workshop are published in ACM Ada Letters, Volume XXXII, Number 1, April 2013.

application-level solutions that may obfuscate the resulting code.

The group agrees that the above considerations provide sufficient motivation to define a new profile, which goes beyond Ravenscar and, quite possibly, includes all of its features.

Discussion ensues on the general characteristics of the profile. Joyce Tokar comments that the idea of a new profile is interesting and attractive but it is hard to tell how difficult it may be to implement. Michael González argues that we should pay attention to allowing features that at the time of Ravenscar were known to be extremely complex to implement. He observes that alternative models may exist, such as e.g., ARINC 653, which might be an interesting target to specify a language profile against. Joyce Tokar voices agreement to that consideration, but also reckons that discussing an ARINC 653 profile would stray the discussion away from the intended focus.

The discussion then moves to the need for dynamic priorities. The question before the group is what language features we really need to be able to suspend / resume individual tasks? The choice is between dynamic priorities (D.5) and asynchronous task control (D.11).

At the end of this first round of high-level discussion, there is full consensus from the group that we need a new profile, distinct from Ravenscar. What we want is a consistent, cohesive profile, and not a string of optional additions to Ravenscar. We should however pay attention to keeping the implementation and certification distance from Ravenscar affordable for language implementers. The group's consensus is also that the starting point for the definition of a new profile should be a clear application programming model, from which we can then determine the implementation requirements, language restrictions, and obstacles to certification. Burns and Wellings' paper [1] is a good starting point.

At this point the Chair invites the group to delve deeper into the discussion of specific features, with special attention at how they would work in a multicore environment.

The first feature on the list is Set_CPU (D.16.1). Including it in the profile supports error recovery strategies that use load balancing. In the envisioned model, tasks are statically allocated to groups, but they can be moved across cores. This feature provides key support to a task splitting approach to handling timing faults whereby overrun-work may be performed opportunistically, via load balancing, on a core different from the one in which the offending task was initially assigned.

Michael González voices his preference for a suspension-only model to one that also allows overrunning tasks to resume. An inconclusive discussion then follows on the implementation complexity that may be incurred by asynchronous task suspension. This issue, among others, needs to be further studied and should become a topic of investigation for IRTAW-16.

The second feature on the list is the restriction "one CPU Timer per task". The rationale for that restriction is that, on a single core, the timer resides in the same CPU as the task to which it belongs. It is natural to extend this notion to multicore. The problem with multicores however is that systems may exist where there is a single clock for all cores: in that case it may become complex to map time events to the CPU where the task resides. The reason to attach the event handler to the CPU where the task executes is that this assignment warrants that if the handler executes then certainly the task does not, and consequently we don't need complex asynchronous control to hold the task from running.

The issue of setting affinities for the handlers of timing events, CPU timers and "normal" interrupt handlers is discussed. The idea is to require the affinity for time-related handlers to be consistent (i.e., equal) to that of the task that causes handler invocation. The discussion then delves into the general problem of what affinities – also for Ravenscar and full Ada – can be set to protected handlers in multicores. It rapidly becomes evident that this is a large, general problem which should be set aside for later and deeper analysis, for discussion at IRTAW-16.

The next feature submitted to discussion is the support for Group Budget: the concept is that fault containment strategies for collections may offer greater flexibility than for individual tasks (yet at the cost of not knowing what the offending task actually is). The group's concern is that a non-trivial implementation burden may be incurred in supporting that feature. To make things simpler we need to assume that the group budget must be per CPU. Intense discussion takes place on whether dynamic group management (i.e., adding and removing tasks from groups at run time) should be preferred to having static group membership only.

The discussion returns to the issue of dynamic priorities: there is majority sentiment that the current slant of the dynamic priority package is too general for our needs. We do not really want any generalized agent to be allowed to set task priorities: we rather want specialized / dedicated handlers (which perhaps could be identified by some restriction) to do that. One way around this problem is to prefer asynchronous task control (with hold only) to dynamic priorities. The group's sentiment on the alternative appears to be divided. The Chair calls for a straw poll, which shows 10 in favour of asynchronous task control, and 8 doubtful abstentions.

The final feature for discussion in the part of the session is entry queues. We still require single entry (for the same reasons why we had that restriction in Ravenscar), but we want to allow multiple calls to queue. We feel the guard should continue to be a Boolean only. 'Count can however be used in the entry body. On that account, an intense yet inconclusive discussion takes place on the safeness, for

our purposes, of the semantics of Hold being called when the task has a call in an entry queue. As part of this slot, the issue is also briefly discussed as to whether nested protected subprogram calls should be allowed in multicore processing: the group observes that the theory of real-time systems has not yet developed a convincing model for nested resources. It may therefore be wiser at this time to disallow nesting.

Before calling this discussion to an end, the Chair invites suggestions for further features of interest for the envisioned profile; the group response includes: barriers in multicores; relative delay statement and relative timing events. The intent is to record this need, invite the group to investigate it in the future, and then discuss the findings at IRTAW-16.

3 Ravenscar and distribution

The Chair then invites a short report on the progress of the University of Cantabria's (UC) group in the development of a Ravenscar- and SPARK-compliant implementation of the Distributed Systems Annex. Héctor Pérez, on behalf of UC, explains that two language features are needed by the current implementation but conflict with the SPARK restrictions: generics; and abstract types.

The group sentiment in that respect is that too strict adherence to SPARK may defeat the purpose of the UC project: "educated" generics and abstract types are useful abstractions for the project and they should be retained.

A side issue was raised before the discussion on this topic came to an end: language support for initialization-level configuration is desired (which comes handy for, e.g., end-point receivers) that does not resort to full-fledged programmatic interfaces, which could be exposed to erroneous usage. No conclusion is reached on this point, other than recording it for further investigation.

4 Code archetypes and programming frameworks

The subsequent slot of discussion focuses – in a joint fashion – on hearing a report on the progress of the work conducted at the University of Padova (UPD) for the finalization of Ravenscar code patterns for automated code generation [3], and at the Universitat Politècnica de València (UPV) for the extension to multicore of the real-time programming framework [4].

Both reports show good and interesting progress. Seeing some complementarity between the qualities of the two respective approaches, the group encourages the teams at UPV and UPD to investigate the possibility of integrating their results.

5 Profiles: Ravenscar and EDF

The final slot of the day's discussion is devoted to examining Alan Burns' proposal in [5] for an EDF version of the Ravenscar Profile. The author's rationale for the proposal is that Ada 2005 supports EDF and mixed dispatching policies via Baker's stack resource protocol [6]. However, the resulting protocol is complicated to understand as well as to implement.

In the author's vision, an interesting alternative to support resource sharing under EDF is to make protected execution non-preemptive. The consequences of that approach are: in the pro side, an easy implementation at language level since priorities are no longer needed and a single ready queue is required; on the cons side instead: longer blocking time for tasks owing to non-preemption during protected execution.

The group's sentiment on the proposal gets quickly divided. For some, Burns' model seems attractive and, in a way, conducive to a Ravenscar adoption of EDF symmetrical, for simplicity and theory support, to Fixed Priority Scheduling (FPS). For others instead, and for Michael González in particular, EDF alone should be considered insufficient for safely programming HRT systems: in Michael's view one would need additional features, either EDF+FPS (as in previous publications from our community) or budget control. Intense discussion takes place on this interesting subject, but it comes to no final conclusion owing to lack of time. The Chair invites the group to continue investigating this matter with a view to reporting progress at IRTAW-16.

References

[1] A. Burns, A.J. Wellings and A.H. Malik (2013), *TTF-Ravenscar: A Profile to Support Reliable High-Integrity Multiprocessor Ada Applications*, Ada Letters.

[2] T. Anderson and P.A. Lee (1990), *Fault Tolerance Principles and Practice*, Prentice-Hall International, 2nd edition.

[3] M. Panunzio and T. Vardanega (2013), *Charting the evolution of the Ada Ravenscar code archetypes,* Ada Letters.

[4] S. Saez, J. Real, and A. Crespo (2013), *Adding Multiprocessor and Mode Change Support to the Ada Real-Time Framework*, Ada Letters.

[5] A. Burns (2013), *An EDF Run-Time Profile based on Ravenscar,* Ada Letters.

[6] T.P. Baker (1991), *Stack-based scheduling of real-time processes*, Journal of Real-Time Systems, 3(1).

Session Summary: Concurrency Issues*

Chair: Juan Antonio de la Puente

Rapporteur: Stephen Michell

1 Introduction

This session was the final session of International Real Time Ada Workshop 15. It was chaired by Juan Antonio de la Puente. Stephen Michell was the rapporteur. Papers and issues discussed were

- Concurrency and real time vulnerabilities under consideration by ISO/IEC/JTC 1/SC 22/WG 23 Programming Language Vulnerabilities Working Group.
- Execution time accounting as being implemented by the Ada programming language and considerations for multiprocessor environments
- Discussion of Set_CPU and deferment of attribute setting in multiprocessor environments.

Section 2 discusses the concurrency vulnerabilities with 2.1 discussing the vulnerabilities in [6] and 2.2 discussing real time vulnerability outlines presented by Stephen Michell at the workshop. Section 3 summaries the discussion of execution time accounting. Section 4 is Ada real time and virtualization.

2 Concurrency Vulnerabilities

2.1 Methodology

As discussed in [2] and [6], the work being done by WG 23 to date does not address the real issues of vulnerabilities presented by concurrent programs. [6] addresses this with a proposal for six vulnerabilities for

- Thread activation,
- Thread termination – directed
- Thread termination – premature termination,
- Shared data access,
- Concurrent data corruption, and
- Concurrency protocol errors.

The workshop examined these proposals and then went further to consider three proposals to develop real time vulnerability writeups for

- Real time timing vulnerabilities,
- Real time thread control, and
- Real time scheduling.

2.2 General Vulnerability Discussion

Steve introduced the topic by first discussing vulnerabilities, the notion of exploits of vulnerabilities. For a general writeup on vulnerabilities, their effects, and the role that programming languages can play in creating and helping to avoid vulnerabilities, see section 5 of [8]. More discussions on weaknesses and vulnerabilities can be found at the Open Web Application Security Project [7], the Common Weakness Evaluations [4], Common Attack Pattern Enumeration [3], and the Build Security In project [1].

There are two kinds of vulnerabilities discussed by TR 24772. The first is called "Language Vulnerabilities" and are documented in section 6 of that document. The second is "Application Vulnerabilities" in section 7 of this document. TR 24772 explains the difference as follows.

"This Technical Report focuses on a particular class of vulnerabilities, language vulnerabilities. These are properties of programming languages that can contribute to (or are strongly correlated with) application vulnerabilities—security weaknesses, safety hazards, or defects. An example may clarify the relationship. The programmer's use of a string copying function that does check length may be exploited by an attacker to place incorrect return values on the program stack, hence passing control of the execution to code provided by the attacker. The string copying function is the language vulnerability and the resulting weakness of the program in the face of the stack attack is the application vulnerability. The programming language vulnerability enables the application vulnerability. The language vulnerability can be avoided by using a string copying function that does set appropriate bounds on the length of the string to be copied. By using a bounded copy function the programmer improves the predictability of the code's execution.

The primary purpose of this Technical Report is to survey common programming language vulnerabilities; this is done in Clause 6. Each description explains how an application vulnerability can result. In Clause 7, a few additional application vulnerabilities are described. These are selected because they are associated with language weaknesses even if they do not directly result from language vulnerabilities. For example, a programmer might have stored a password in plaintext (see [XYM]) because the programming language did not provide a suitable library function for storing the password in a non-recoverable format."

* The Proceedings of the 15th International Real-Time Ada Workshop are published in ACM Ada Letters, Volume XXXII, Number 1, April 2013.

The workshop spent most of the session discussing the real time issues that could become vulnerabilities. They are identified here as presented by Stephen on slides to lead the session.

It should be noted that TR 24772 uses the term "thread" where Ada uses "task" to designate entities that can execute concurrently. For the purpose of this section, the term thread will be used exclusively.

2.3 General Concurrency Vulnerabilities

2.3.1 Thread Activation [CGA]

Steve presented the general principals as presented in [6] section 5.1 Thread Activation [CGA]. There was general agreement that there are language issues involved in the creation of threads such as resource exhaustion, undetected failure to activate of some threads, and the resulting system failures that can occur during creation. Therefore, this vulnerability belongs in section 6 Programming Language Vulnerabilities of ISO IEC TR 24772. There were no recommendations for further subdivision of the vulnerability or consolidation with other vulnerabilities. No further application workarounds or programming language extensions were discussed.

2.3.2 Thread Termination (on request) [CGT] (and premature) [CGS]

The workshop decided to discuss the issues of thread termination together. Both vulnerabilities were presented as being appropriate for section 6 of ISO IEC TR24772 as programming language vulnerabilities, since, even if the language does not have a concurrency component, the underlying environment as presented by its libraries have such a paradigm and can expose the issues. Languages that have concurrency as part of the language must consider all termination issues. There were no recommendations for further subdivision of the vulnerability or consolidation with other vulnerabilities.

Joyce raised the issue that finalization of data and of control space after the directed termination of a thread is an issue that must be discussed in [CGT]. Also, there must be a way to detect attempts to interact with threads that have been terminated or requested to terminate, both in the period of termination and after the thread has been terminated. These issues will be added to the submissions to WG 23.

2.3.3 Concurrent Data Access [CGX]

Stephen noted that this vulnerability was split from [CGY] so that issues of direct access to data in concurrent and real time data can be separated from shared resources that happen in external components of the system such as filing systems, environment variables, and databases.

The workshop reviewed the writeup, and agreed that this was a language vulnerability and as such belonged in section 6 of TR 24772. Andy noted that the issue of "volatile" of a shared variable was missing, in the sense that languages often reorder reads and assignments and may not be aware that other assignment operations exist in other threads. The "volatile" directive notifies the language processor not to perform such re-orderings, and to make fetches and assignments as atomic as possible.

It was noted that there is another recommendation to application developers to always specify data elements as "volatile" to prevent language processors from performing such reordering.

2.3.4 Concurrent Data Corruption [CGY]

Steve proposed in his presentation to target this vulnerability to TR 24772 sect 7 as an application vulnerability, since external resources outside of the application itself, and the concurrent components accessing it may not be part of the same program, so language-specific mechanisms to control access are not feasible in general. There were no proposed additional issues, language vulnerabilities or application workarounds discussed.

2.3.5 Concurrent Protocol Errors [CGM]

The workshop discussed the vulnerability's target to TR 24772 sect 6 as a language or OS vulnerability and agreed that it was appropriate because a number of languages and most operating systems (through libraries) provide protocols for concurrency paradigms. No significant issues were identified as missing. No other application workarounds or language avoidance mechanisms were identified.

2.4 Real Time Vulnerability Discussion

Stephen led the discussion from slides prepared to present three real time vulnerabilities, with the plan to use the workshop discussions to generate writeups suitable for WG 23.

2.4.1 Real Time Timing [CGQ]

The application vulnerability was identified as an application vulnerability (as opposed to a programming language vulnerability) since almost all real time timing issues arise as a result of hardware issues, low level kernel issues and application issues. The main issues identified initially were

- Drift between clocks, such as real time and time of day clocks, or between clocks on different processors
- Failure to track time of day clock updates due to leap seconds, time zone changes or corrections; and
- mismatches between time accounting and notification requests, such as posted wakeup times or deadlines

The workshop added to this

- Rollover of bounded clocks can cause timer calculations to fail
- mismatches between the resolution of the clock and the expectations of the application can cause wakeups or deadlines to be too late or too early (usually too late)
- transfers between a time of day clock and real time clock can result in loss of precision and missed deadlines or missed wakeups

The effects of these errors can be

- Wrong calculations
- Missed deadlines
- Wakeups that are too early or too late causing portions of the concurrency structure to fail
- guardian code (that relies on being notified if deadlines are missed) may misbehave if timers drift

Recommended approaches for application developers are:

- Develop systems whose concurrency model is amenable to static analysis, such as model checking,
- Perform analysis of all aspects of timing with as much rigour as is possible;
- Choose a language or environment that provides the capabilities to express time, select the appropriate time paradigm, and clock management mechanisms
- Implement measures to monitor time accounting, and drift between clocks, and be prepared to take corrective action
- Identify mechanisms to identify misbehaving systems such as heartbeats or watchdog timers; and
- Include monitors with the ability to reset the complete system back to a known state before continuing.

The usual result of such errors is erroneous behaviours or misbehaving feedback systems. Systems that rely on on-time calculations and events and that experience such erroneous behaviour may experience catastrophic failures. Arbitrary code execution is unlikely.

2.4.2 Real Time Thread Control

Burns and Wellings [2] identified real time thread control issues as another real time potential vulnerability. The workshop agreed that this was already covered in the vulnerability Protocol Lock Errors [CGM].

2.4.3 Real Time Scheduling [CGP]

Real Time Scheduling was identified as a programming language vulnerability since there are a few languages and operating systems that provide scheduling protocols suitable for real time, and use of the high level paradigms, such as the Priority Ceiling Protocol can fix many of the issues in this domain. On the other hand, to make real time systems function to specification, even using these protocols requires applications designers that clearly understand the domain and take the necessary steps to allocate attributes to threads (such as deadlines and priorities) and to communications and protocol management code.

The main issues identified were

- Priority inversion when a thread of a lower urgency (lower priority or later deadline) is executing and preventing one of higher urgency (that is ready to execute) from executing.
- Scheduling mistakes resulting in threads not completing their work in the required time
- Missed thread deadlines due to priority inversions, scheduling mistakes and protocol errors
- Missed interrupts or events
- Excessive input or events
- Excessive use of a resource
- Lack of access policy or queuing policy resulting in indefinite waiting for some threads

These can result in

- Missed system deadlines
- Complete system failure
- System halting
- System ceasing to process input or deliver output
- Instability
- System not meeting specifications

It was noted that, in the real time realm, that even seemingly small changes, such as a change of a single priority to a single thread can have very large impacts.

Recommended approaches for application developers are:

- Develop systems whose concurrency model is amenable to static analysis, such as model checking,
- Perform this analysis with as much rigour as is possible;
- Choose a language or environment that provides the capabilities of priority inheritance and priority ceiling protocol, and to use those capabilities;
- If using multiprocessors, be aware of the issues with scheduling, thread interaction and data and cache coherency across multiple processors and be very conservative in the assignment of threads to processors;
- Identify mechanisms to identify misbehaving systems such as heartbeats or watchdog timers; and
- Include monitors with the ability to reset the complete system back to a known state before continuing.

Errors in this domain can be used to create covert channels, and can lead to complete system shutdown or misbehaviour, but arbitrary code execution as a result of this vulnerability alone is unlikely.

3 Execution-time accounting of interrupts

This session followed on from Session A, where the focus of the discussion was specialization to multiprocessor environments..

While the workshop was in progress, participants received an advanced copy of a paper by Kristoffer Gregersten on the implementation of execution-time accounting of interrupts. A paper submitted by the same author [5] was instrumental in leading IRTAW 14 to recommend that Ada implement this accounting as part of Ada 2012, which is now being finalized and which includes this capability. In the paper to be submitted to the Ada Users Journal, Kristoffer summarizes an implementation of this new capability in the MARTE real time Ada executive on a single processor embedded processor. In his paper he documents a number of tests that show negligible additional overhead to support this accounting and explains the reasons for this.

Jose Ruiz reported that AdaCore has similar functionality implemented for GNAT Ada on embedded Leon32 and PowerPC processors and suggested that the capability was very useful for their clients.

3.1 Deferred attribute setting mechanism

Sergio raised the issue that the current wording in the draft Ada 2012 reference manual says that System.Multiprocessors.Dispatching_Domains.Set_CPU can cause excessive context switches for tasks that are executing in a protected operation on CPU1 but make a call to Set_CPU to move itself to CPU 2. Clause D.16.1(27) of the ARM says

"A call of Set_CPU is a task dispatching point for task T. If T is the Current_Task the effect is immediate, otherwise the effect is as soon as practical."

This could be interpreted to mean that Set_CPU calls into the kernel, effectively suspends the task while it moves it to "CPU", possibly causing at least 2 context switches. It was noted that the same wording exists for System.Multiprocessors.Dispatching_Domains.Add_Task.

Here is a model of what can happen:

- T1 is executing on CPU1 with priority P1.
- T2 is executing on CPU2 with priority P2>P1.
- T1 calls a protected operation PO_A that has a ceiling priority of P3>P2>P1. Within PO_A, T1 (which is now executing at priority P3) executes a Set_CPU or a Set_Task call to assign itself to CPU2.
- T2 is immediately moved to CPU2 while still executing inside PO_A and preempts T2.
- P2 either is switched to a different cpu (depending upon T2's affininty) or is delayed until P1 completes execution of PO_A.
- T1 completes PO_A, and its priority drops to P1, causing another context switch to permit T2 to resume execution.

There was significant discussion on the various issues. It was noted that there is no significant issue if task T1 is blocked or delayed and another task changes its CPU or affinity. Changes to the running task, however, can have the difficulties described above. In a real time system where T2 has a deadline, such additional overhead of context switches and execution of protected operations in place of T2's execution could cause T2 to miss its deadline.

Miguel raised the issue that more deferred operations are required, such as Delay_Until_And_Set_Deadline (See ARM D.2.6), and noted that any combination of Set_CPU, Set_Deadline, and Set_Priority can cause excessive context switches and priority inversions (such as the one described above) unless we can explicitly defer the setting of these operations. Miguel proposed the following subprograms:

```
Ada.Dispatching.EDF.Set_Next_Deadline(
      D : in Deadline;
      T : in Ada.Task_Identification.Task_Id
       := Ada.Task_Identification.Current_Task);

System.Multiprocessors.Dispatching_Domains.
Set_Next_Task( CPU : in CPU_Range;
                  T   : in Task_Id := Current_Task);

Ada.Dynamic_Priorities.Set_Next_Priority(
     Priority : in System.Any_Priority;
     T        : in Ada.Task_Identification.Task_Id
           := Ada.Task_Identification.Current_Task);
```

Miguel proposed to add the following subprogram that set the deferred attributes immediately:

```
Set_Attributes(
      Attr : Deferred_Attributes;
      T : in Ada.Task_Identification.Task_ID :=
               Current_Task);
```

Where

```
type Deferred_Attributes is record
      CPU: CPU_Range;
      D: Deadline;
      ...
end Deferred_Attributes;
```

The advantage of this approach would be that it provides a consistent Set_Attributes, whereas the calls

```
Set_Next_Deadline, Set_Next_CPU, ...,
```

permits different task calls to interleave, creating different "Next" attributes to be set, at the point of the Set_-Attributes.

There was general support for this concept. Steve proposed that these be in a child package of each page, possibly called something like Deferred. It was not clear where Set_Attributes would be declared.

There was discussion as to what would happen if Set_Now was not called. The presumption was that the deferred settings would happen when

- Set_Attributes was called
- The task awoke from a delay, delay_until,

- The task was removed from a suspension object,
- The task was released from a protected entry, after completing the operation of that entry
- The task completed a protected operation, or the outermost protected operation if it was executing in a nested protected operation.

It was noted that one cannot rely upon a task being suspended, blocked or delayed for a significant period of time after having a deadline, priority or affinity set, hence there needs to be a subprogram call to set the attributes immediately.

Implementation approaches were discussed. The approach of having room for deferred attributes in the Task Control Block or task attributes was discussed. Such attributes would become effective when the task was released from a delay or blockage, or immediately upon a call to Set_Attributes.

It was agreed that these issues needed further investigation, modelling and trial implementations before a formal proposal could be made to WG9.

4 Ada Real-Time Services and Virtualization

There was a discussion of Ada real time services and virtualization. It was decided that this was not a language issue, hence discussion terminated.

5 Wrap up and Conclusions

This session was the final one of the workshop. The work on vulnerabilities gave Stephen material to feed back into the work of WG 23. He expects that the next publishing of TR 24772 will contain at 6-8 concurrency vulnerabilities based on the work done here.

The next meeting of the workshop is planned for York area, UK in the spring of 2013.

References

[1] Build Security In website, www.BuildSecurityIn.us-cert.org.

[2] Burns, Alan and Wellings, Andy (2010), *Language Vulnerabilities - Lets Not Forget Concurrency*, Proceedings of Real Time Ada Workshop 14, ACM SIGAda Letters, Volume 30 Issue 1.

[3] Common Attack Pattern Enumeration and Classification project web site, www.capec.mitre.org

[4] Common Weakness Enumeration web site, www.cwe.mitre.org

[5] Gregertsen, Kristoffer (2010), *Execution-time control for interrupt handling*, Proceedings of IRTAW 14, ACM SIGAda Letters., vol. 30 Issue 1.

[6] Michell, Stephen (2011), *Programming Language Vulnerabilities – Proposals to Include Concurrency Paradigms*, Transactions of IRTAW 15, ACM SIGADA Letters, Volume 31 Issue 4.

[7] Open Web Application Security Project, www.owasp.com.

[8] ISO IEC TR 24772 (2010), *Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use*, International Standards Organisation.

Ada Conference UK 2013

Sophie Robinson, AdaCore, 46 rued'Amsterdam, Paris 75009, France

Abstract

The Ada Conference UK 2013, operated by the Centre for Software Reliability (CSR), took place this year on 25th April at the IET Birmingham: Austin Court Conference Centre.

1 Introduction

The 25th April was a significant date in the Ada calendar as it was the date of the 2013 edition of the Ada Conference UK, the biennial event showcasing industrial and academic uses of the Ada programming language. This year was specifically significant due to the recent release of Ada 2012.

Following three previous editions of the event, the organisers this year decided to move the event north to the IET's Austin Court, Birmingham, a historical building on the canal-side location of Brindley Place.

Delegates were drawn by the success of previous years, the programme of speakers and the launch of the new Ada programming language, Ada 2012. The day started with a comical introduction from the ever-brilliant John Barnes and included an excellent half-day tutorial from Tucker Taft, who had flown over from the USA. The highlight of the day's activities had to be the outstanding talks of the keynote speakers, John C. Knight and Robert Dewar. John presented the talk "Ada Types – Are They Sufficient?" and Robert concluded the day with his talk, "I'm as Mad as Hell, and I'm Not Going To Take This Anymore!"

John Barnes summed up the event nicely, stating "I much enjoyed the Ada conference recently held in Birmingham; it is always a pleasure to meet users and enthusiasts (both old and new) within a convivial location. We were privileged to learn about the technical strides being made by Ada and SPARK from experts such as Tucker Taft and Stuart Matthews. As well as a number of talks on various aspects of high integrity applications, our keynote speakers were John Knight who emphasised the importance of getting our types right, and Robert Dewar who ended the day by cajoling us all to attack the prevalence of relaxed attitudes to software glitches."

The technical track offered talks on a diverse range of subjects by leading industrial experts. Parallel to the technical track, there was a stream of (technically oriented) vendor talks from Atego, Altran, AdaCore, Abstract Solutions, Wind River and Vectorcast.

2 Abstracts from the talks that made up the technical track

Ada Types–Are They Sufficient?

John Knight, University of Virginia, Charlottesville.

Abstract: In this talk I will introduce a type system based on general real-world semantics. The type system permits any attributes of real-world entities to be encoded in the type of their machine representations, together with type rules that allow checks to be based on the real-world behavior of the typed entities. As well as checking for violations of real-world type rules in expressions, the real-world type system allows checking of crucial properties in systems of systems. For example, inconsistent states that might arise in an avionics system between the coordinates being used in different component sub-systems can be detected statically.

Proposed Revision to Def Stan 00-56.

Paul Caseley, dstl.

Abstract: Def Stan 00-56 defines the contractor requirements for system safety for MOD Defence equipment. This presentation will provide insight to the direction and content of the draft Def Stan 00-56 issue 5 which proposes new concepts that will influence system and software safety related projects. Additionally, an insight to the direction of thinking behind the proposed programmable electronics standard, Def Stan 00-55 issue 3, will also be discussed.

Ada 2005 in Practice.

Jeff Cousins, BAE Systems.

Abstract: This presentation discusses what code changes we had to make for Ada 2005, and what additional changes we chose to make to utilise Ada 2005 features. Although some Ada 2005 features have proved useful, use of them has been limited as legacy code would generally have already found some solution using Ada 95.

Vendor extensions and tools are then discussed. Although vendor extensions are generally avoided for portability

reasons, a few are so useful that they have tempted us. A first look is then taken at using Ada 2012, evaluated using the vendor's compiler.

From the model to the target to certification– trends in growing use of code from model based developmental systems in high integrity environments.

Ian Harry, LDRA.

Abstract: This presentation will discuss the trend in which an increasing portion of embedded code in high integrity environments is directly derived from models. This trend has been driven, in part, due to high-fidelity modeling and code generation tools. These tools are extremely powerful in making sure that embedded code matches the model, even in cases where the model and the application are rapidly changing. However, in safety critical environments these tools are not, in and of themselves, adequate for system verification. They must be paired with embedded target-verification tools and matched with appropriate process standard goals.

We will discuss how tools can automate best practices to track code from the model to the target, ensuring that verification tasks such as code coverage are completed correctly. In addition, requirements traceability capabilities ensure that all of the elements are adequately connected. This is particularly important with process standards such as DO-178C, which require specific model based elements to be connected to specific verification tasks and results. This workflow will be discussed in depth in the context of both DO-178C and other process standards.

Applying D0333/DO178C–a white paper.

Nick Tudor, D-RisQ Ltd.

Abstract: This presentation will discuss the crucial steps in using DO333, the Formal Methods Supplement to DO178C/DO278A. It provides a view of what that document considers to be 'Formal Methods' and how to use them to claim credit for certification. While some detail from the document will be discussed, it is – of course – not intended to provide a tutorial on the entire document! A white paper from D-RisQ, upon which the presentation is based, will be made available on request.

"I'm as Mad as Hell, and I'm Not Going To Take This Anymore!"

Robert Dewar, New York University and AdaCore.

Abstract: We live in the age of the computer "glitch" where disastrous errors in all kinds of software system – dismissed as glitches – have horrible consequences, including loss of life from defective medical equipment, major financial disruption from malfunction of financial software, accidental release of dangerous prisoners etc. It's time we didn't tolerate this unacceptable state of affairs. In this talk we will speculate on what could be done to fix this problem and, in particular, how we can negotiate a marriage (or at least a civil union) between the worlds of testing and formal proof in the search for solutions.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/adaanswers/gems.

Gem #144: A Bit of Bytes: Characters and Encoding Schemes

Emmanuel Briot, AdaCore

Abstract. This Gem describes some of the concepts behind character encoding and Unicode. It explains why multiple character sets exist, and how to deal with them in your application if you want to handle international input and output.

Let's get started…

This Gem starts with a problem. As a French native, I often manipulate text files that contain accented letters (those accents, by the way, were often introduced as a shorthand to replace letters in words, to save paper when it was still an expensive commodity). Unfortunately, depending on how the file was created, my programs do not necessarily see the same byte contents (which depends on the encoding and the character set of the file), and, if I just try to display them on the screen (either in a text console, or in a graphical window), the output might not read like what I initially entered.

Glyphs

At this point, let's introduce the notion of glyphs. These are the visual representations of characters. For instance, I want "e-acute" to look like an "e" with a small acute accent above it. This visual representation is the final goal in a lot of applications, since that's what the user wants to see. In other applications, however, the glyphs are irrelevant. For instance, a compiler does not care how characters are displayed on your screen. It needs to know how to split a sequence of characters into words, but that's about it. It assumes your console, where error messages are displayed, will display the same glyphs you had in your source file when given the same bytes as the source file itself.

A text file does not embed the description of what its representation looks like. Instead, it is composed of bytes, which are combined in certain ways (sometimes called character encoding schemes) to make up code points. These code points are then matched to a specific character using a character set. Finally, the font determines how the character should be represented as a glyph.

A character's exact representation (its glyph) really depends on the font you are using, since a "lower case a" might have widely different aspects that depend on the font. This is outside the scope of this Gem, though.

In general, your application is not concerned with the mapping of characters to glyphs via the font. This is all taken care of by either the text console, or the GUI toolkit you are using. Your application will often let the user choose her preferred font, and then make sure to pass valid characters. The toolkit does the complex work of representing the characters. For example, this work is the role of the Pango toolkit (accessible from GtkAda).

Character Sets

A repertoire is a set of generally related characters, for instance the alphabets used to spell English or Russian words.

A character set is a mapping from a repertoire to a set of integers called code points. A given character, as we shall see, might exist in several different character sets with different code points.

Most of the standard character sets (sometimes abbreviated as charsets) are specific to one language. For instance, there exist ISO-8859-1 (also known as Latin-1) and ISO-8859-15, which are used for West European languages; we also have ISO-8859-5 and KOI8-R, which are different, but both used for Russian; Windows introduced a number of code pages, which are in fact character sets specific to that platform; Japanese texts often use ISO-2022-JP, whereas Chinese has several standard sets.

Let's take the simplest of them all, the ASCII charset. Most developers are familiar with it. For instance, in this set the code point 65 is associated with the letter upper-case-A. This set includes 128 characters, 31 of which have no visual representation. It contains no accented letters, but is basically appropriate for representing English texts.

In a lot of Western European languages, like French, ASCII was not sufficient, so ISO-8859-1 was built on top of it. The first 128 characters are the same, so code point 65 is still upper-case-A. But it also adds 128 extra characters, for instance 233 is lower-case-e-with-acute. See the Wikipedia page on ISO-8859-1 for more details.

Another example is ISO-8859-5, for Russian text, which is incompatible with ISO-8859-1, although it is also based on ASCII. So 65 is still upper-case-A, but this time 233 is cyrillic-small-letter-shcha and lower-case-e-with-acute does not exist.

As a result, if an application is reading an ISO-8859-5 encoded file, but believes it is ISO-8859-1, it will display an invalid glyph for most of the Russian letters, obviously making the text unreadable for the user.

In most applications (for instance, the GPS IDE), there is a way to specify which character set the application should expect the files to be encoded in by default, and a way to override the default encoding for specific files.

There exists one character set that includes all characters that exist in all the other character sets (or at least is meant to), and this is Unicode (somewhat akin to ISO-10646). It includes thousands upon thousands of characters (and more are added at each revision), while avoiding duplicates. For compatibility with a lot of existing applications, the first 256 characters are the same as in ISO-8859-1, so upper-case-A is still 63, and lower-case-e-with-acute is still 233. But now cyrillic-small-letter-shcha is 1097.

Nowadays, a lot of applications (and even programming languages) will systematically use Unicode internally. For instance, the GTK+ graphic toolkit only manipulates Unicode

for internal strings, and so does Python 3.x. So whenever a file is read from disk by GPS, it is first converted from its native character set to Unicode, and then the rest of the application no longer has to care about character sets.

Given the size of Unicode, there are few (if any) fonts that can represent the whole set of characters, but that's not an issue in general since most applications do not need to represent Egyptian hieroglyphs...

Another major part of the Unicode standard is a set of tables to find the properties of various characters: which ones should be considered as white space, how to convert from lower to upper case, which letters are part of words, etc. This knowledge is often hard-coded in our applications and often involves a major change when an application decides to use Unicode internally.

Character Encoding Schemes

We now know how to represent characters as a combination of code points and a character set. But we often need to store those characters in files, which only contain bytes. That seems relatively easy when the code point is less than 256, but becomes much less obvious for other code points, like the 1097 we saw earlier.

In practice, this issue is solved in a number of ways. Encoding schemes such as the Japanese ISO-2022-JP use a notion of plane shift: special bytes indicate that from now on the bytes should be interpreted differently, until the next plane shift. Decoding and encoding therefore requires knowledge of the current state.

Unicode itself defines three different encoding schemes (with their variants), which are known as UTF-8, UTF-16, and UTF-32. The last number indicates the number of bits that each character is encoded in. Therefore, in UTF-32, each character occupies four bytes, which allows the whole set of Unicode characters to be represented. Decoding and encoding is therefore trivial, but there is a major waste of space associated with UTF-32.

In UTF-16, each character is encoded in two bytes, which is enough for all characters used by spoken languages. Other characters are for specific usage, like Egyptian hieroglyphs. For code points that do not fit in two bytes, Unicode defines a few special bytes (the surrogate pairs) that are similar to the plane shifts we described earlier. Thus, there is much less wasted space, but decoding and encoding becomes a bit more complex.

The above two encoding schemes are not backward compatible: an application that was written before Unicode and that only knew about ASCII or ISO-8859-1 will not understand the input strings properly.

For this reason, and to save even more space, Unicode also defines the UTF-8 encoding. For all ASCII characters, they are still represented as before using a single byte. Characters greater than 127 are encoded as a sequence of several bytes (and it is guaranteed that all bytes but the last are not part of ASCII).

Properly manipulating a UTF-8 string requires the use of specialized routines (since moving forward one character means moving forward 1 to 6 bytes). However, a casual application can, for instance, skip to the next white space character as it did before by moving forward one byte at a time and stopping when it sees 32 (a space) or 13 (a newline). This property can often be used by applications that do not need to represent the characters, like the example of the compiler we mentioned at the beginning.

Although the notions of character sets and character encoding schemes are orthogonal, often these notions are conflated. For instance, when someone mentions ISO-8859-1, it usually means the character set as well as its standard representation, where each character is represented as a single byte. Likewise, someone talking about UTF-8 will typically mean the Unicode character set together with the UTF-8 character encoding scheme.

Conversions

We now have almost all of the pieces in place, except for the conversion between character sets. In theory, it is enough to decode the input stream using the proper character encoding scheme, then find the mapping for the code points from the origin to the target character set, and finally use the target encoding scheme to represent the characters as bytes again.

When a character has no mapping into the target character set (for instance the e-acute in the Russian iso-8859-5), the application needs to decide whether to raise an error, ignore the character, or find a transliteration (for example, using e' for e-acute).

This is obviously tedious, and requires the use of big lookup tables for all the character sets your application needs to support.

On Unix systems, there exists a standard library, iconv, to do this conversion work on your behalf. The GNU project also provides such an open-source library for other systems.

We have recently added a binding to this library in the GNAT Components Collection (GNATCOLL.Iconv), making it even easier to use from Ada. For instance:

```
with GNATCOLL.Iconv;   use GNATCOLL.Iconv;
procedure Main is
      EAcute : constant Character :=
                                 Character'Val(16#E9#);
      -- in ISO-8859-1

      Result : constant String := Iconv
               ("Some string " & EAcute,
                To_Code   => UTF8,
                From_Code => ISO_8859_1)
begin
   null;
end Main;
```

XML/Ada has also included such conversion tables for a while, but supports many fewer character sets. Check the Unicode.CSS.* packages.

As you can see above, we are reusing the string type, since, in Ada, a string is not associated with any specific character set or encoding scheme. In general, as we mentioned before, this is not an issue, since an application will use a single encoding internally (UTF-8 in most cases). Another approach is to use a Wide_String or Wide_Wide_String. The same comment as for UTF-16 and UTF-32 applies: these make character manipulation more convenient, but at the cost of wasted memory.

Manipulating UTF-8 and UTF-16 strings

The last piece of the puzzle, once we have a Unicode string in memory, is to find each character in it. This requires specialized subprograms, since the number of bytes is variable

for each character.

XML/Ada's Unicode module includes such a set of subprograms in its Unicode.CES.* packages. In general, going forward is relatively easy and can be done efficiently, whereas going backward in a string is more complex and less efficient.

The GNAT run-time library also contains such packages, for instance GNAT.Encode_UTF8_String and GNAT.Decode_UTF8_String. In particular, the latter provides Decode_Wide_Character, Next_Wide_Character, and Prev_Wide_Character, to find all the characters in a string.

Gem #146: Su(per)btypes in Ada 2012 - Part 1

Yannick Moy, AdaCore

Abstract. The new revision of Ada is full of features for specifying properties of types. In this series of three Gems, we describe three aspects that can be used to state invariant properties of types. This first Gem is concerned with the Static_Predicate aspect.

Let's get started…

Ada 2012 is full of features for specifying a rich set of type properties. In this series of three Gems, we describe three aspects that can be used to state invariant properties of types and subtypes. This first Gem is concerned with the Static_Predicate aspect.

Static_Predicate can be specified on scalar types and subtype definitions to state a property that all objects of the subtype must respect at all times. Take for example a type Day representing the days of the week:

```
type Day is (Monday, Tuesday, Wednesday, Thursday,
      Friday, Saturday, Sunday);
```

To state that T_Day is the (sub)type of days whose name starts with a 'T', we can write:

```
type T_Day is new Day with Static_Predicate => T_Day
      in Tuesday | Thursday;
```

or

```
subtype T_Day is Day with Static_Predicate => T_Day
      in Tuesday | Thursday;
```

Now the compiler will warn about a program that assigns a value statically known to be different from Tuesday or Thursday to a T_Day object.

We'll proceed with using the second definition above. For example, on this incorrect code:

```
D : T_Day := Day'First; -- Incorrect
```

GNAT generates the following warning at compile time:

```
>>> warning: static expression fails static predicate
check on "T_Day"
```

The compiler also checks the completeness of case expressions and case statements involving T_Day arguments. For example, on this code:

```
case D is
  when Tuesday => ...
  when Friday => ...  -- Incorrect
end case;
```

GNAT generates the following errors at compile time:

```
>>> missing case value: "Thursday"
>>> static predicate on "T_Day" excludes value "Friday"
```

If Friday is replaced by the correct value Thursday, then the code compiles quietly.

Finally, the compiler generates run-time checks for any erroneous write of a day other than Tuesday or Thursday in an object of type T_Day, which makes it easy to detect violations of the predicate of a type as soon as it occurs! Note that to enable run-time checking of Static_Predicate (and other kinds of assertions specified by aspects) it's necessary to compile with the switch -gnata (or else enable assertion checking with the pragma Assertion_Policy).

For example, suppose we have a procedure Next that advances its argument to the next day, and we want to define a similar procedure, Next_T, that advances its argument of subtype T_Day. Here's the definition of procedure Next:

```
procedure Next (D : in out Day) is
begin
  if D = Sunday then
    D := Monday;
  else
    D := Day'Succ (D);
  end if;
end Next;
```

Following is a failed attempt at defining Next_T:

```
procedure Next_T (D : in out T_Day) is
begin
  Next (D); -- Incorrect
  while D not in T_Day loop
    Next (D);
  end loop;
end Next_T;
```

Let's add a test of this code:

```
with Days; use Days;
procedure Main is
  D : T_Day := Tuesday;
begin
  Next_T (D);
end Main;
```

When this code is compiled with assertions enabled (-gnata) and run, it issues a run-time error:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE :
Static_Predicate failed at days.adb:3
```

This points to the first line where Next is called in Next_T. Indeed, on entry to Next_T, the value of D is Tuesday, so Next returns Wednesday, which does not satisfy the Static_Predicate of T_Day, but is assigned to a T_Day, hence triggering a run-time error. The correct version of Next_T uses a temporary variable of type T_Day'Base, which strips off all constraints from T_Day, including the predicate if present:

```
procedure Next_T (D : in out T_Day) is
  Tmp : T_Day'Base := D;
begin
  Next (Tmp);
  while Tmp not in T_Day loop
    Next (Tmp);
```

```
   end loop;
   D := Tmp;
end Next_T;
```

In the next Gem in this series we'll see how to use a related aspect called Dynamic_Predicate.

Gem #147: Su(per)btypes in Ada 2012 - Part 2

Yannick Moy, AdaCore

Abstract: In the previous Gem in this series, we saw how the aspect Static_Predicate can be used to state properties of scalar objects that should be respected at all times. This Gem is concerned with the Dynamic_Predicate aspect.

Let's get started…

The previous Gem in this series showed how the aspect Static_Predicate can be used to state properties of scalar objects that should be respected at all times. This Gem is concerned with the Dynamic_Predicate aspect, which can be used on all type and subtype declarations (not just scalar ones).

Consider for example a type Message encoding the dates when a message was sent and received, where dates are represented by strings, such as "1789-07-14" for the fourteenth of July 1789:

```
type Day is new String (1 .. 10);

type Message is record
   Sent     : Day;
   Received : Day;
end record;
```

To state that a message reception date should always be greater than the date it was sent, we can write:

```
type Message is record
   Sent     : Day;
   Received : Day;
end record with
  Dynamic_Predicate => Message.Sent <=
                       Message.Received;
```

Note that the type name itself is used as a prefix of the components named in the predicate. In this context the name of the type denotes what Ada calls the current instance of the type, which at run time will denote the actual object the predicate is applied to.

In contrast to Static_Predicate, the compiler cannot determine in general if a Dynamic_Predicate will fail, so it inserts run-time checks at certain required locations in the code:

- when assigning to a variable of the subtype
- when passing an input parameter of the subtype
- when returning an output parameter to an object of the subtype
- when converting a value to the subtype

For example, on the following incorrect code:

```
M : Message := (Received => "1776-07-04", Sent =>
"1783-09-03");  -- ncorrect
```

Compiling it with assertions and running it leads to the following error:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE :
Dynamic_Predicate failed at main.adb:3
```

If the values of the Sent and Received components are corrected to reflect the actual event ordering of the proclamation of the Independence of the United States and the date of the treaty of Paris ending the American Revolutionary War, then the generated code executes without errors.

Beware that no run-time checks are inserted when assigning to individual components, so the predicate can be silently violated between assignments and calls. For example, if the definition above separately assigns each component of M, even if the value for Received and Sent are appropriately ordered:

```
   M : Message;  -- incorrect
begin
   M.Received := "1783-09-03";  -- incorrect
   M.Sent := "1776-07-04";      -- predicate is correct here
```

This code does not lead to a run-time failure, but if we pass the message before it is completely initialized to some procedure Process taking it as input parameter:

```
procedure Process (M : Message);
```

Compiling the resulting code with assertions and running it again leads to an error:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE :
Dynamic_Predicate failed at main.adb:7
```

Note that Dynamic_Predicate is more flexible than Static_Predicate: it can be applied to more forms of types and more general predicate expressions. For example, the mod operator is not allowed outside a static expression in a Static_Predicate, so the type of odd numbers must be defined with a Dynamic_Predicate:

```
subtype Odd is Integer with Dynamic_Predicate =>
                            Odd mod 2 = 1;
```

Likewise, a user function can be called in a Dynamic_Predicate, but not in a Static_Predicate.

GNAT conveniently provides an aspect Predicate that can be used anywhere a Dynamic_Predicate is allowed, and analyzes it as a Static_Predicate when possible.

In the next and final Gem in this series on type and subtype contracts we'll look at a related aspect called Type_Invariant.

Gem #148: Su(per)btypes in Ada 2012 - Part 3

Yannick Moy, AdaCore

Abstract: In the previous two Gems of this series, we saw how the aspects Static_Predicate and Dynamic_Predicate can be used to state properties of objects that should be respected at all times. This Gem is concerned with the Type_Invariant aspect.

Let's get started...

In the previous two Gems, we saw how aspects Static_Predicate and Dynamic_Predicate can be used to state properties of objects that should be respected at all times. This third and final Gem in the series is concerned with an aspect called Type_Invariant.

The Type_Invariant aspect can be used with private types, to define a property that all objects of the types should respect outside of the package where the types are declared. Take for example a type Communication storing the messages between various parties, based on the Message type used in the previous Gem:

```
package Communications is
  type Message_Arr is array (Integer range <>) of
       Message;
  type Communication (Num : Positive) is private;
private
  type Communication (Num : Positive) is record
    Msgs : Message_Arr (1 .. Num);
  end record;
end Communications;
```

To state that messages should be ordered by date of reception, we can add the aspect to the full type:

```
type Communication (Num : Positive) is record
  Msgs : Message_Arr (1 .. Num);
end record with
 Type_Invariant =>
  (for all Idx in 1 .. Communication.Num-1 =>
      Communication.Msgs(Idx).Received <=
      Communication.Msgs(Idx+1).Received);
```

The compiler will insert run-time checks to ensure that this property holds at prescribed locations in the code:

- at object initialization (including by default!)
- on conversions to the type
- when returning an object from a public function defined in the type's package
- on out and in out parameters, when returning from a public procedure of the type's package

For example, consider the following incorrect code that fails to initialize Com to a correct value satisfying the invariant:

```
Com : Communication (2);  -- incorrect
```

Compiling it with assertions and running it leads to the following error:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE :
failed invariant from communications.ads:16
```

But if we give the object an explicit value, through a creation function Create defined in unit Communications, then the object declaration is elaborated without errors:

```
Coms : Communication (2) := Create (A);
```

Inside the Create function, the initialization of Coms must respect the invariant, but after that, the invariant could be violated between the time Coms is declared, and the time it is returned.

```
function Create (A : Message_Arr)
      return Communication is
  Coms : Communication :=
       (Num => A'Length, Msgs => A);
begin
  -- statements before the return might violate the
  -- invariant
  return Coms;
end Create;
```

Ada requires that the type invariant be checked on every part of a parameter that has type Communication, where a part can be a component of a record, or an element of an array, or any such combination. For example, it is checked on every element of the array returned by Create_N or potentially modified by Update_N:

```
type Communication_Arr is array (Integer range <>)
      of Communication;
function Create_N return Communication;
procedure Update_N (A : in out Communication_Arr);
```

Importantly, the invariant is not checked on subprograms declared in the private part or in the package body. These subprograms are internal operations, and should be callable on objects whose invariant does not hold. Likewise, the invariant is not checked on parameters of mode in, for example on query functions used in the definition of the type invariant itself. This is fortunate, since otherwise this would easily cause infinite loops!

As a side note, it's worth mentioning that GNAT also provides an aspect with the name Invariant, which is a synonym for the Type_Invariant aspect (and implemented before Type_Invariant appeared in Ada 2012).

This Gem ends the series of three Gems on su(per)btypes in Ada. Together with Static_Predicate and Dynamic_Predicate, Type_Invariant provides new ways to state properties of your data, both in new and existing programs, so try them out!.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU 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