Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, NY 10121-0701

Ada LETTERS

Volume XXXII Number 2 August 2012

Table of Contents

A Publication of SIGAda,
the ACM Special Interest Group on Ada

ACM Digital Library

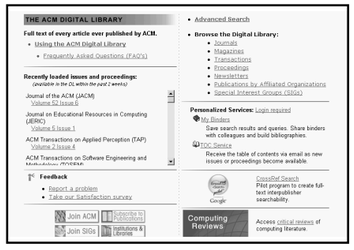
www.acm.org/dl

The Ultimate Online INFORMATION TECHNOLOGY Resource!

Powerful and vast in scope, the **ACM Digital Library** is the ultimate online resource offering unlimited access and value!

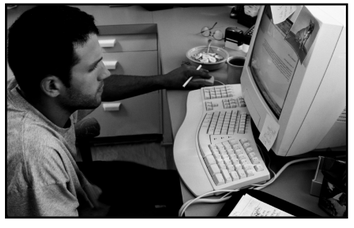
The **ACM Digital Library** interface includes:

• **The ACM Digital Library** offers over 45 publications including all ACM journals, magazines, and conference proceedings, plus vast archives, representing over two million pages of text. The ACM DL includes full-text articles from all ACM publications dating back to the 1950s, as well as third-party content with selected archives. **www.acm.org/dl**



• **The Guide to Computing Literature** offers an enormous bank of more than one million bibliographic citations extending far beyond ACM's proprietary literature, covering all types of works in computing such as journals, proceedings, books, technical reports, and theses! **www.acm.org/guide**

• **The Online Computing Reviews Service** includes reviews by computing experts, providing timely commentary and critiques of the most essential books and articles.



Available only to ACM Members.
Join ACM online at **www.acm.org/joinacm**

To subscribe to the ACM Digital Library, contact ACM Member Services:

Phone: 1.800.342.6626 (U.S. and Canada)
+1.212.626.0500 (Global)
Fax: +1.212.944.1318
Hours: 8:30 a.m.-4:30 p.m., Eastern Time
Email: acmhelp@acm.org
Mail: ACM Member Services
General Post Office
PO Box 30777
New York, NY 10087-0777 USA

ACM Professional Members can add the ACM Digital Library for only $99 (USD). Student Portal Package membership includes the Digital Library. Institutional, Corporate, and Consortia Packages are also available.

Association for Computing Machinery

Advancing Computing as a Science & Profession

AD10

join today!

SIGADA & ACM

www.acm.org/sigada **www.acm.org**

The **ACM Special Interest Group on Ada Programming Language** (SIGAda) provides a forum on all aspects of the Ada language and technologies, including usage, education, standardization, design methods, and compiler implementation. Among the topics that SIGAda addresses are software engineering practice, real-time applications, high-integrity & safety-critical systems, object-oriented technology, software education, and large-scale system development. SIGAda explores these issues through an annual international conference, special-purpose Working Groups, active local chapters, and its *Ada Letters* publication.

The **Association for Computing Machinery** (ACM) is an educational and scientific computing society which works to advance computing as a science and a profession. Benefits include subscriptions to *Communications of the ACM*, *MemberNet*, *TechNews* and *CareerNews*, full and unlimited access to thousands of online courses and books, discounts on conferences and the option to subscribe to the ACM Digital Library.

- ❑ SIGAda (ACM Member) ..... $ 25
- ❑ SIGAda (ACM Student Member & Non-ACM Student Member) ..... $ 10
- ❑ SIGAda (Non-ACM Member) ..... $ 25
- ❑ ACM Professional Membership ($99) & SIGAda ($25) $124
- ❑ ACM Professional Membership ($99) & SIGAda ($25) & ACM Digital Library ($99) $223
- ❑ ACM Student Membership ($19) & SIGAda ($10) ..... $ 29
- ❑ *Ada Letters* only ..... $ 53
- ❑ Expedited Air for *Communications of the ACM* (outside N. America) ..... $ 56

payment information

Name _____

ACM Member # _____

Mailing Address _____

_____

City/State/Province _____

ZIP/Postal Code/Country _____

Email _____

Mobile Phone _____

Fax _____

Credit Card Type: ❑ AMEX ❑ VISA ❑ MC

Credit Card # _____

Exp. Date _____

Signature _____

Make check or money order payable to ACM, Inc

ACM accepts U.S. dollars or equivalent in foreign currency. Prices include surface delivery charge. Expedited Air Service, which is a partial air freight delivery service, is available outside North America. Contact ACM for more information.

Mailing List Restriction
ACM occasionally makes its mailing list available to computer-related organizations, educational institutions and sister societies. All email addresses remain strictly confidential. Check one of the following if you wish to restrict the use of your name:

- ❑ ACM announcements only
- ❑ ACM and other sister society announcements
- ❑ ACM subscription and renewal notices only

Questions? Contact:
ACM Headquarters
2 Penn Plaza, Suite 701
New York, NY 10121-0701
voice: 212-626-0500
fax: 212-944-1318
email: acmhelp@acm.org

Remit to:
ACM
General Post Office
P.O. Box 30777
New York, NY 10087-0777

SIGAPP

www.acm.org/joinsigs

Association for Computing Machinery

Advancing Computing as a Science & Profession

Volume XXXII Number 2, August 2012

Table of Contents

A Publication of SIGAda,
the ACM Special Interest Group on Ada

For advertising information contact:
Advertising Department
2 Penn Plaza, Suite 701, New York, NY 10121-0701
Phone: (212) 869-7440; Fax (212) 869-0481

Is your organization recognized as an Ada supporter? Become a SIGAda INSTITUTIONAL SPONSOR! Benefits include having your organization's name and address listed in every issue of Ada Letters, two subscriptions to Ada Letters and member conference rates for all of your employees attending SIGAda events. To sign up, contact Rachael Barish, ACM Headquarters, 2 Penn Plaza, Suite 701, New York, NY 10121-0701, and email: MEETING@ACM.ORG, Phone: 212-626-0603.

Interested in reaching the Ada market? Please contact Jennifer Booher at Worldata (561) 393-8200 Ext. 131, email: platimer@worldata.com. Please make sure to ask for more information on ACM membership mailing lists and labels.

Ada Letters (ISSN 1094-3641) is published three times a year by the Association for Computing Machinery, 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA. The basic annual subscription price is $20.00 for ACM members.

POSTMASTER: Send change of address to Ada Letters:
ACM, 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA

From the Editor's Desk

Alok Srivastava

Welcome to this issue of ACM Ada Letters. In this issue you will find very interesting paper "Making Alive Register Transfer Level and Transaction Level Modeling in Ada" by Negin Mahani. In this paper Negin has shown use of a specialized form of Ada as a system description language, like the way SystemC is used for description of systems. The paper discusses the specification of proposed SystemAda, its hardware description style, its RTL link, and description of a TLM 1.0 channel using SystemAda.

The issue also provides details on AdaCore compiled **Ada Gems:**

- Code Archetypes for Real-Time Programming - Part 4 by *Marco Panunzio*
- Reference Counting in Ada - Part 1 by *Emmanuel Briot*
- High Performance Multi-core Programming - Part 2 by *Pat Rogers*
- Reference Counting in Ada - Part 2: Task Safety by *Emmanuel Briot*
- Reference Counting in Ada - Part 3: Weak References by *Emmanuel Briot*
- SOAP/WSDL server part by *Pascal Obry*
- SOAP/WSDL client part by *Pascal Obry*
- Code Archetypes for Real-Time Programming - Part 5 by *Marco Panunzio*
- Gprbuild and Configuration Files - Part 1by *Johannes Kanig*
- Lady Ada Kisses Python - Part 1 by *Emmanuel Briot*
- Lady Ada Kisses Python - Part 2 by *Emmanuel Briot*

In its last ACM SIGAda Extended Executive Committee (EEC) meeting held in Denver during SIGAda 2011 conference, the EEC discussed a possible reorganization of the Executive Committee to include fewer elected officer positions to be consistent with several other SIGs. Therefore the SIGAda By-laws have been amended to reduce the number of elected SIGAda officers from six to four: Chair, Vice Chair, Secretary-Treasurer and International Representative. The justification for this change is that it will reduce the number of volunteers that SIGAda needs to find to fill the officer positions. It also enables other volunteers to serve in Conference Chair and Local Arrangements chair positions to support the SIGAda annual conferences.

Also in this issue ACM SIGAda Chair Ricky Sward has provided the SIGAda Annual Report for July 1, 2011 - June 30, 2012 stating accomplishments and the challenges.

Here you will find the announcement of High-Integrity Language Technology SIGAda 2013 conference to be held next year from mid-October to mid-November in Pittsburgh, Pennsylvania USA. Other major Ada events, the 16th International Real-Time Ada Workshop - IRTAW 2013 will be held from 17-19 April 2013 at Kings Manor, York, England and the 18th International Conference on Reliable Software Technologies - Ada-Europe 2013 will take place in Berlin, Germany, from June 10 to 14, 2013.

Regular contributor Trudy Levine has provided listing of sources for reusable software components.

Ada Letters is a great place to submit articles of your experiences with the language revision, tips on usage of the new language features, as well as to describe success stories using Ada. We'll look forward to your submission. You can submit either a MS Word or Adobe PDF file (with 1" margins and no page numbers) to our technical editor:

Pat Rogers, Ph.D.
AdaCore, 207 Charleston, Friendswood, TX 77546 (USA), +1 281 648 3165, rogers@adacore.com

We look forward to hearing from you!

Alok Srivastava, Ph.D.
Technical Fellow, TASC Inc.
475 School St, SW; Washington, DC 20024 (USA), +1 202 314 1419 Alok.Srivastava@TASC.Com

Editorial Policy (from Alok Srivastava, Managing Editor)

As the editor of Ada Letters, I'd like to thank you for your continued support of ACM SIGAda, and encourage you to submit articles for publication. In addition, if there is some way we can make Ada Letters more useful to you, please let me know. Note that Ada Letters is now on the web! See http://www.acm.org/sigada/ada_letters/index.html. The two newest issues are available only to SIGAda members. Older issues beginning March 2000 are available to all.

Ada is standing on its own merits, lots of people and organizations have stepped up to provide new tools, mechanisms for compiler validation/assessment, and standards (especially ASIS). The Ada 2005 language version is fulfilling the market demand of robust safety and security elements and thereby generating a new enthusiasm into the software development and the same is expected from the incoming Ada 2012. Ada Letters is a venue for you to share your successes and ideas with others in the Ada community. Be sure to take advantage of it so that we can all benefit from each other's learning and experience.

As some of the other ACM Special Interest Group periodicals have moved, Ada Letters also transitioned from quarterly to a **tri-annual publication**. With exception of special issues, Ada Letters now is going to be published three times a year, with the exception of special issues. The revised schedules and submission deadlines are as follows:

Deadline	**Issue**	**Deadline**	**Issue**
October 1st, 12	December, 2012	February 1st, 13	April, 2013
June 1st, 13	August, 2013	October 1st, 13	December, 2013

Please send your article to Dr. Pat Rogers at rogers@adacore.com

Guidelines for Authors

Letters, announcements and book reviews should be sent directly to the Managing Editor and will normally appear in the next corresponding issue.

Proposed articles are to be submitted to the Technical Editor. Any article will be considered for publication, provided that topic is of interest to the SIGAda membership. Previously published articles are welcome, provided the previous publisher or copyright holder grants permission. In particular, keeping with the theme of recent SIGAda conferences, we are interested in submissions that demonstrate that "Ada Works." For example, a description of how Ada helped you with a particular project or a description of how to solve a task in Ada are suitable.

Although Ada Letters is not a refereed publication, acceptance is subject to the review and discretion of the Technical Editor. In order to appear in a particular issue, articles must be submitted far enough in advance of the deadline to allow for review/edit cycles. Backlogs may result in an article's being delayed for two or more issues. Contact the Managing Editor for information on the current publishing queue.

Articles should be submitted electronically in one of the following formats: MS Word (preferred) Postscript, or Adobe Acrobat. All submissions must be formatted for US Letter paper (8.5" x 11") with one inch margins on each side (for a total print area of 6.5" x 9") with no page

numbers, headers or footers. Full justification of text is preferred, with proportional font (preferably Times New Roman, or equivalent) of no less than 10 points. Code insertions should be presented in a non-proportional font such as Courier.

The title should be centered, followed by author information (also centered). The author's name, organization name and address, telephone number, and e-mail address should be given. For previously published articles, please give an introductory statement (in a distinctive font) or a footnote on the first page identifying the previous publication. ACM is improving member services by creating an electronic library of all of its publications. Read the following for how this affects your submissions.

Notice to Contributing Authors to SIG Newsletters:

By submitting your article for distribution in this Special Interest Group publication, you hereby grant to ACM the following non-exclusive, perpetual, worldwide rights:

- to publish in print on condition of acceptance by the editor
- to digitize and post your article in the electronic version of this publication
- to include the article in the ACM Digital Library
- to allow users to copy and distribute the article for noncommercial, educational or research purposes

However, as a contributing author, you retain copyright to your article and ACM will make every effort to refer requests for commercial use directly to you.

Notice to Past Authors of ACM-Published Articles

ACM intends to create a complete electronic archive of all articles and/or other material previously published by ACM. If you have a work that has been previously published by ACM in any journal or conference proceedings prior to 1978, or any SIG Newsletter at any time, and you do NOT want this work to appear in the ACM Digital Library, please inform permissions@acm.org, stating the title of the work, the author(s), and where and when published.

Back Issues

Back issues of Ada Letters can be ordered at the price of $6.00 per issue for ACM or SIGAda members; and $9.00 per issue for non-ACM members. Information on availability, contact the ACM Order Department at 1-800-342-6626 or 410-528-4261. Checks and credit cards only are accepted and payment must be enclosed with the order. Specify volume and issue number as well as date of publication. Orders must be sent to:

ACM Order Department, P.O. Box 12114, Church Street Station, New York, NY 10257 or via FAX: 301-528-8550.

KEY CONTACTS

Technical Editor
Send your book reviews, letters, and articles to:

Pat Rogers
AdaCore
207 Charleston
Friendswood, TX 77546
+1-281-648 3165
Email: rogers@adacore.com

Managing Editor
Send announcements and short notices to:

Alok Srivastava
TASC Inc.
475 School Street, SW
Washington DC 20024
+1-202-314-1419
Email: Alok.Srivastava@TASC.Com

Advertising
Send advertisements to:

William Kooney
Advertising/Sales Account Executive
2 Penn Plaza, Suite 701
New York, NY 10121-0701
Phone: +1-212-869-7440
Fax: +1-212-869-0481

Local SIGAda Matters
Send Local SIGAda related matters to:

Greg Gicca
AdaCore
1849 Briland Street
Tarpon Springs, FL 34689, USA
Phone: +1-646-375-0734
Fax: +1-727-944-5197
Email: Gicca@AdaCore.Com

Ada CASE and Design Language Developers Matrix
Send ADL and CASE product Info to:

Judy Kerner
The Aerospace Corporation
Mail Stop M8/117
P.O. Box 92957
Los Angeles, CA 90009
+1-310-336-3131
Email: kerner@aero.org

Ada Around the World
Send Foreign Ada organization info to:

Dirk Craeynest
c/o K.U.Leuven, Dept. of Computer Science,
Celestijnenlaan 200-A, B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be

Reusable Software Components
Send info on reusable software to:

Trudy Levine
Computer Science Department
Fairleigh Dickinson University
Teaneck, NJ 07666
+1-201-692-2000
Email: levine@fdu.edu

SIGAda Working Group (WG) Chairs

See http://www.acm.org/sigada/ for most up-to-date information

Ada Application Programming Interfaces WG
Geoff Smith
Lightfleet Corporation
4800 NW Camas Meadows Drive
Camas, WA 98607
Phone: +1-503-816-1983
Fax: +1-360-816-5750
Email: gsmith@lightfleet.com

Ada Semantic Interface Specification WG
http://www.acm.org/sigada/wg/asiswg/asiswg.html
Bill Thomas
The MITRE Corp
7515 Colshire Drive
McLean, VA 22102-7508
Phone: +1-703-983-6159
Fax: +1-703-983-1339
Email: BThomas@MITRE.Org

Education WG
http://www.sigada.org/wg/eduwg/eduwg.html
Mike Feldman
420 N.W. 11th Ave., #915
Portland, OR 97209-2970
Email: MFeldman@seas.gwu.edu

Standards WG
Robert Dewar
73 5th Ave.
New York, NY 10003
Phone: +1-212-741-0957
Fax: +1-232-242-3722
Email: dewar@cs.nyu.edu

Ada Around the World
(National Ada Organizations)

From: http://www.ada-europe.org/members.html

Ada-Europe
Tullio Vardanega
University of Padua
Department of Pure and Applied Mathematics
Via Trieste 63
I-35121, Padova, Italy
Phone: +39-049-827-1359
Fax: +39-049-827-1444
E-mail: tullio.vardanega@math.unipd.it
http://www.ada-europe.org/

Ada-Belgium
Dirk Craeynest
C/o K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200-A, B-3001
Leuven (Heverlee), Belgium
Phone: +32-2-725 40 25
Fax : +32-2-725 40 12
E-mail: Dirk.Craeynest@cs.kuleuven.be
http://www.cs.kuleuven.be/~dirk/ada-belgium/

Ada in Denmark
Jørgen Bundgaard
E-mail: Info at Ada-DK.org
http://www.Ada-DK.org/

Ada-Deutschland
Peter Dencker, Steinackerstr. 25
D-76275 Ettlingen-Spessartt, Germany
E-mail: dencker@parasoft.de
http://www.ada-deutschland.de/

Ada-France
Association Ada-France
c/o Jérôme Hugues
Département Informatique et Réseau
École Nationale Supérieure des Télécommunications, 46, rue Barrault
75634 Paris Cedex 135, France
E-mail: bureau@ada-france.org
http://www.ada-france.org/

Ada Spain
J. Javier Gutiérrez
P.O. Box 50.403
E-28080 Madrid, Spain
Phone: +34-942-201394
Fax : +34-942-201402
E-mail: gutierjj@unican.es
http://www.adaspain.org/

Ada in Sweden
Rei Stråhle
Box Saab Systems
S:t Olofsgatan 9A
SE-753 21 Uppsala, Sweden
Phone: +46-73-437-7124
Fax : +46-85-808-7260
E-mail: Rei.Strahle@saabgroup.com
http://www.ada-i-sverige.se/

Ada in Switzerland
Ahlan Marriott
White Elephant GmbH
Postfach 327
CH-8450 Andelfingen, Switzerland
Phone: +41 52 624 2939
Fax : +41 52 624 2334
E-mail: ada@white-elephant.ch
http://www.ada-switzerland.org/

Italy
Contact: tullio.vardanega@math.unipd.it

Ada-Europe Secretariat
e-mail: secretariat@ada-europe.org

Making Alive Register Transfer Level and Transaction Level Modeling in Ada

Negin Mahani
Electrical and Computer Engineering Department, Faculty of Engineering, Campus #2
University of Tehran, 14399 Tehran, IRAN
negin@cad.ece.ut.ac.ir
negin_mahani@yahoo.com

Abstract

Over the past 50 years, design of hardware has evolved from transistor level to register transfer level (RTL), and now to transaction level. Transaction Level Modeling (TLM) enhances simulation performance of today's complex digital systems and also provides the ability of early design space exploration. TLM divides a system into computation parts, i.e. processing elements, and communication parts, i.e. channels and sockets. The inherent concurrency of Ada along with its object oriented features gives it potentials for being used as a TLM language.

In this paper, we use a specialized form of Ada as a system description language, like the way SystemC is used for description of systems. We refer to our form of Ada as SystemAda and we use a public Ada compiler (Gnat) to evaluate system descriptions written in Ada. SystemAda is meant for modeling system behavior and structure at the transaction level and we consider possible approaches for extending Ada to meet these requirements. This paper discusses the specification of our proposed SystemAda, its hardware description style, its RTL link, and description of a TLM 1.0 channel using SystemAda.

1. Introduction

To cope with the complexity of electronic systems, high level description languages have evolved for describing electronic hardware at system level. Today, Transaction Level Modeling has established itself as a common way of describing digital systems at the system level for design and architecture exploration. TLM allows designers to focus on the functionality of the design and not to be concerned with low level (RTL) details. At this level of abstraction, a hardware system is divided into processing elements and communication channels. At the top level, this system consist of a structural interconnections of processing elements, where interconnections become channels and sockets instead of wires at the gate level, or buses at RTL [1].

For the design of such systems, hardware description languages that are different than those used in today's register transfer languages should be used. The focus of such languages should be on processing elements connected by channels, instead of registers and logic units connected by buses as in register transfer level languages. Furthermore, languages for system level descriptions must be capable to easily interface with software languages, to enable designers to design complex hardware/software systems in one environment [1].

Our study of Ada language features, compilers, and support for concurrency indicates that this language is a good candidate for describing hardware at RT and transaction levels. Inspired by an existing RTL package for Ada, we have developed an RTL link for Ada that provides system level hardware designers with sufficient RT level description capabilities. In addition, we have developed a package containing a basic transaction level channel to provide Ada system level designers with hardware description features found in today's transaction level hardware description languages. The combination of our added RTL and transaction level features makes our SystemAda that is described in this paper.

In addition to providing constructs for covering hardware at certain level, a hardware description language at any level has to provide a minimum set of constructs for describing hardware at its immediate next lower level of abstraction. Therefore, we have to cover some preliminary RT level constructs for creating a transaction level hardware language from Ada. This work focuses on TLM while providing a sufficient link to RTL.

In this paper, we introduce SystemAda as a system description language for hardware systems. The work is partitioned based on two distinct concepts: RTL and TLM. In the RTL concept, we explain different options for linking Ada to RTL and expand one of them. In the

TLM concept, we describe *tlm_fifo* as the most basic TLM channel which will be used for describing other channels. The rest of this paper is organized as follows. Section 2 briefly introduces Transaction Level Modeling and TLM standards released by OSCI. Section 3 contains an overview of Ada as a Hardware Description Language (HDL). In Section 4, major requirements for SystemAda are explained. Section 4.1 introduces existing methods for linking Ada to RTL and adopts one such method and adds several new features to it. Section 4.2 contains an overview of TLM channels and their implementation in Ada. Section 5 describes *tlm_fifo* application in a *master-slave* architecture using Ada *tasks*. In section 6 we have implemented a Network on chip system usisg our ada packages. Finally, in Section 7 we present the conclusion of this work.

2. Transaction Level Modeling

Recent advances in semiconductor technology enables the designers to integrate hundreds of embedded processors on a single chip which is called System on Chip (SoC). This fast-paced increasing of complexity of digital systems, on the other hand, has introduced new challenges to the design community. An important challenge is to handle the complexity of designing such systems with increased productivity and decreased time-to-market. Electronic System Level (ESL) design methodologies solve this problem by starting the design from higher abstraction levels than RTL. Transaction Level Modeling (TLM) has been proposed as the key step toward ESL methodology. In TLM point of view, a system is divided into two parts: computation parts and communication parts. The focus of the TLM is on communication and this is performed through passing high level data structures called transactions between computation parts.

SystemC that is a library on C++ has been used widely for design at transaction level. Open SystemC Initiative (OSCI) which works on the definition and standardization of SystemC as a system-level language has released two versions of the TLM standard. In TLM 1.0, channels are the basic communication elements, while in TLM 2.0 the concept of the sockets has been introduced as the communication infrastructure.

In this work, we will examine the possibility of using another language, i.e. Ada, as a TLM language. In general, having the link to RTL and the ability to describe TLM channels are two essential requirements of a TLM language. The inherent concurrency of Ada along with its object oriented features gives it potentials for satisfying these requirements. We will discuss these issues in later sections.

3. Ada as an HDL

Most hardware designers use an HDL such as VHDL or Verilog for their RT level descriptions, C or C++ for software parts of their designs, and SystemC and its TLM derivation for fast simulation of system level designs. With SigAda's project [2], RTL design, software parts of a design, and software for design test and verification can all be done in the same language and environment.

The idea of using Ada as an HDL goes back to 1980's. At that time, Ada compilation was slow and hardware design abstraction was at gate level, making Ada not the best alternative for hardware description. In 1981, Ada was used as a base language for the development of VHDL [3]. In 1995, a new version of Ada called Ada95 was defined with object oriented features that could be used for high abstraction level design such as TLM [4]. Considering Ada as the VHDL base language, a popular HDL for complex hardware description at RTL, it has the proper root to be used for system level description as a TLM language.

Because hardware systems comprise activities that are performed concurrently, an important requirement for a hardware description language is its support for concurrency. In addition to concurrency, an HDL for transaction level modeling should have some language features for object oriented modeling, genericity, and abstract communication.

Considering language features and structures of Ada, we can easily conclude that the TLM standard is implementable in this language. In addition to its capabilities for transaction level modeling, Ada has links with the lower level, i.e. RTL. The VHDL language that has been derived from Ada is similar to Ada in syntax. VHDL hardware modules that are described with entity-architecture parts are equivalent to the specification-body structures of the Ada language. Such observations led us to conclude that Ada, as it is presently defined, has potentials for description of hardware.

From the linguistics point of view, Ada compilers have the advantage of being strongly typed and being thorough in the compilation process such that most syntax and semantic errors are detected during compilation. Such features make Ada a language that is more feasible for design error detection than C/C++, their additional patches, or their SystemC derivations [5][6].

Furthermore, Ada2005 has services which provide the ability to trigger events at specific times, invoking threads, and facilitating process synchronization. Object oriented programming could be linked to real time activities using these capabilities [5][6].

4. Major requirements for SystemAda

In addition to supporting TLM abstraction level, system level languages like SystemC also provide a strong link to RTL. In general, having the link to RTL and the ability to describe TLM channels are two essential requirements of a TLM language. In this work, we consider the main concepts of the OSCI TLM 1.0 standard [7][8], and base our requirements on the main features supported by this standard.

Since communication is the main focus of the TLM and all TLM channels have been defined based on *fifo* channel (*tlm_fifo*), we have developed a *tlm_fifo* channel in this paper. We show how this channel is instantiated and used for interconnection of processing elements.

As mentioned above, a hardware description language needs a link to its most immediate lower level language; in case of our system level, the lower level is RTL. Therefore, we need to provide a link between our system level SystemAda and an existing RTL hardware description language.

4.1. Linking Ada to RTL

There are several options for linking Ada to RTL. The following sections describe the different options. Our RTL part of Ada is based on the method described in the last section. We have created an HDL package for Ada using this method.

4.1.1. Linking using Ada pragmas. An option for creating an HDL link for Ada is using the ability of Ada to communicate with a language which supports RTL. The most useful *pragma*s are *import, export,* and *convention* [9].

According to Ada95 language reference manual, the *import pragma* imports a subprogram from a foreign language into an Ada program. The *export pragma* exports an Ada subprogram to another language, and the *convention pragma* specifies that a certain type should use the storage conventions of a given foreign language. It is also used on subprograms if they are of the callback type [9][10].

Since Ada and C++ programs can interface by Ada *pragma*s (*import, export,* and *convention*), combining Ada and SystemC RTL core is possible.

4.1.2. Linking with VHDL. The second option for providing a link between Ada and RTL is using GHDL. GHDL is an open source VHDL compiler that can simulate Ada programs [11].

Command lines for this compiler provide analyzing VHDL design files, binding the design, and finally compiling the Ada program, binding it and linking it to the VHDL design. Figure 1 shows GHDL and GNAT compiler commands achieving this linking.

```
$ ghdl -a design.vhdl
$ ghdl --bind design
$ gnatmake my_prog -largs `ghdl -listlink
design`
```

Figure 1. GHDL and GNAT compiler commands

4.1.3. Linking by mapping VHDL to ADA. Another possibility for creating a link between Ada and an RT level HDL is mapping VHDL structures to Ada. The U.S. Air Force Research Lab has provided examples of how to describe VHDL constructs in Ada. Signals are unique objects in VHDL and their declarations and assignments are modeled in Ada. VHDL signal attributes have been modeled using an Ada record. This record is called a *signal description record* [12].

In general, the work described in [12] has mapped VHDL language defined types, signal declarations, and signal assignments to various Ada constructs. For example, for implementing VHDL signal assignment an *if* or a *case* statement is used.

4.1.4. Linking by an Ada HDL package. We have created an HDL package for providing a link between Ada and a hardware description language. The basic combinational components of an RT level description consist of word gates (array of gates), multiplexers, decoders, encoders, and arithmetic units. In addition, sequential components are registers and counters (Table 1).

Table 1. Digital logic basic elements

Component Type	Subtype
Elementary logic gates	AND, OR, NAND, NOR, Inverter, and their array versions
Decoder	-
Encoder	-
Multiplexer	-
Full adder	-
Basic latches	SR, D, Gated SR, Gated D
Triggered D flip-flop	-

SigAda's project in 1998 [2] proved that Ada83 could be used as a hardware description language. Since we are using Ada95 to describe hardware at TLM, we have used this version of Ada to develop a package called *UT_Ada_HDL* to provide RTL designers with the required primitives. For this purpose, we have implemented basic RTL elements mentioned above using Ada95 procedures. Based on this and the fact that Ada is an object oriented language, by taking advantage of its reusability facility,

a hardware designer can use Ada procedures to develop more complex elements[16] [17].

UT_Ada_HDL package which is partially shown in Figure 2 contains the Ada script and package made of all the digital logic basic elements shown in Table 1. The basic types and subtypes in this package are as follows:

- Subtype *input*: This type is *Boolean* and is the same as SigAda's input.
- Subtype *output*: This type is *Boolean* and is the same as SigAda's output.
- Type *bus*: This is a *Boolean* array of natural range.
- Type *state*: This is an enumeration type with high and low enumeration elements. This type indicates clock states high and low.
- Type *edge*: This type is a *Boolean* array of natural range. User can use this array to specify clock edges.
- Type *dimension2_bus*: This type is a two dimensional *Boolean* matrix of natural range. This type is introduced for more complex RTL packages.

```
package UT_Ada_HDL is

subtype input  is Boolean;
subtype output is Boolean;

type bus is array
(natural range <>)of Boolean;

type dimension2_bus is array
(natural range<>, natural range<>) of
Boolean;

type state is (high, low);

type edge is array
(natural range<>) of Boolean;

type edge_not is array
(natural range<>) of Boolean;
      …
procedure nand_array (x1: in out bus;
x2: in out bus; n: in out   integer;
xout: out bus);

procedure mux (en:in input;
mux_in: in input; data1: in input;
data2: in input; mux_out: out output);

procedure SR_latch (S: in out input;
R: in out input; Q_a: out output;
Q_b: out output);

procedure fulladder (input1: in input;
input2: in input; carry_in: in Boolean;
carry_out: out output; sum: out output);
      …
end UT_Ada_HDL;
```

Figure 2. UT_Ada_HDL package

Included in this package are procedures *Invert, And_bit, or_bit, nand, nor*, and *mux.* These procedures use Ada's *Boolean* operations for faster simulation. Since word gates inputs use *bus* data type, direct use of *Boolean* operations is not allowed. Therefore, we have used type conversion between *bus* and *Boolean* data types. As an example, the implementation of a *nand* word gate as shown in Figure 3 uses *Boolean* type conversion for *x1* and *x2* input bus types. The invert procedure that also is shown in this figure returns complement of *xouttemp* as *nand* output.

```
procedure nand_array (x1: in out bus; x2: in
out bus; n: in out integer; xout: out bus)
is
xouttemp,xouttemp_not,x1temp,x2temp :
Boolean;
begin
  for i in  reverse x1'range loop
    xouttemp:= Boolean(xout(i));
    x1temp:= Boolean(x1(i));
    x2temp:=Boolean(x2(i));
    xouttemp:= x1temp and x2temp;
    invert(xouttemp,xouttemp_not);
    xout(i):= xouttemp_not;
  end loop;
end nand_array;
```

Figure 3. Nand_array procedure

A simple multiplexer with two data lines, one select line, and an enable control input is shown in Figure 4. As mentioned before, this *mux* is included in our *UT_Ada_HDL* package.

```
procedure mux (en:in input; mux_in: in input;
data1: in input; data2: in input; mux_out: out
output) is
begin
   if en = true then
      if mux_in = true then
         mux_out:= mux_in and data1;
      else
         mux_in_invert:= not mux_in;
         mux_out:= mux_in_invert and  data2;
      end if;
   else
      put("off");
   end if;
end mux;
```

Figure 4. Multiplexer procedure

```
procedure SR_latch (S: in out input; R: in
out input; Q_a: out output; Q_b: out output)
is
begin
    delay 0.004 ;
    Nor(S,Q_a, q_btemp);
    Nor(R,Q_b, q_atemp);
    Q_a:= q_atemp;
    Q_b:= q_btemp;
End SR_latch;
```

Figure 5. SR_Latch procedure

Figure 5 shows implementation of an SR latch. In this procedure, Ada *delay* construct has been used to imply the actual data passing delay in hardware.

4.2. Describing TLM channels using Ada

This section starts with the description of Ada *tasks* that are the main language constructs we have used for describing TLM channels. This will be followed by a subsection for describing the main TLM 1.0 channel, i.e. *tlm_fifo.*

4.2.1. Task overview. *Tasks* are the basic elements for implementing concurrency in Ada. *Task* units can communicate with each other, and each will be in progress for a specified time. The language semantics make it look like that *task*s are running on separate computer systems. *Task*s have this ability using the *entry* concept, which defines what information should be sent to a *task* when it is invoked, and what should be done. *Task*s can be sensitive to activation of one or more *entries*. *Task*s have a declaration part and a body part. A *task* body part describes the functionality of the *task* [13][14][15].

Each *task* can use Ada's *select* block and its *accept* statement. The *select* block, with the *accept* statements inside it, provides alternative entry points for messages into a *task*. The *accept* statements in the *select* block are separated by OR to allow the *task* to accept a call on each entry [13][14][15].

4.2.2. TLM_FIFO channel in Ada. Every TLM 1.0 channel can be implemented based on the *tlm_fifo* channel. This channel is *generic* in size and type. As a result, we define a *generic fifo* channel in Ada to be used later to define other TLM channels.

```
with Ada.Text_IO;
    …
generic type fifo_element is(<>);
PACKAGE fifo IS
  TYPE fifo_node;
  TYPE fifo_pointer IS ACCESS fifo_node;
  TYPE fifo_node IS RECORD
    …
   input : fifo_element;
   output: fifo_element;
    …
  PROCEDURE add_fifo;
  PROCEDURE rem_fifo;
    …
   task type fifo_task is
      entry add;
      entry remove;
      entry stop;
   end fifo_task;
   tlm_fifo : fifo_task;
END fifo;
```

Figure 6. Generic fifo package

The *generic* keyword in Ada has the functionality of the *template* in C++. Figure 6 shows the code of our *fifo* specification that can be used as a TLM channel. This *fifo* is a linked list that has no size limitation. Because of the *generic* element type, the type of its nodes can be of any types.

The *tlm_fifo* of Figure 6 represents a hardware component for communication between several running processes. This component that is used at transaction level, must have concurrency as an essential requirement for modeling hardware systems. For this reason, we have used a *task type,* as the basic unit of concurrency in our *fifo*. The *tlm_fifo task* in this figure is an instance of the *task* type called *fifo-task*.

Our *fifo task* has the *add* and *remove* entries, whose names describe their functionalities. A *generic fifo* provides users with the opportunity to determine the type of its elements. The program shown in Figure 7 shows how we define an integer *fifo* [1].

```
with Ada.Text_IO;
use Ada.Text_IO;
with fifo;
package fifo_channel is new fifo(integer);
```

Figure 7. Generating an integer fifo

5. TLM master-slave architecture

We have modeled a TLM master-slave architecture in Ada using *task types* with two computational modules: *master* and *slave*. The block diagram of the TLM master-slave architecture is shown in Figure 8.

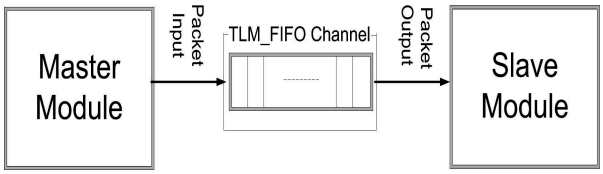


Figure 8. TLM master-slave architecture

```
with fifo_channel;
    …
procedure tlm_master_slave is
   task  slave is
    …
    fifo_channel.tlm_fifo.remove;
    …
   task  master is
    …
       fifo_channel.tlm_fifo.add;
    …
  task simulation is
    …
begin
   simulation.start;
   delay 1.0;
   simulation.stop;
end tlm_master_slave;
```

Figure 9. TLM Master-slave

Figure 9 shows a procedure that defines master and slave interconnections using a *tlm_fifo*. This procedure is composed of three *task*s, each containing an infinite loop inside its body: *master, slave* and *simulation*. These *task*s have two entries *start* and *stop*. The *master task* enters integer numbers into the integer *fifo* by sending *fifo_channel.tlm_fifo.add* message to *tlm_fifo task*. On the other hand, the *slave task* removes integers from the integer *fifo* by sending *fifo_channel.tlm_fifo.remove* message to *tlm_fifo task*, and performs its operation on the obtained data.

In the main block of the program (Figure 9), *simulation.start* massage starts the *simulation task*. After a delay of 1.0, the *simulation.stop* message will stop the *simulation task*. The *simulation task* of Figure 10 by accepting the *start* message enters the loop and sends the *start* massage to *master* and *slave task*s to notify that the simulation has been started. By accepting the *stop* message, the simulation *task* exits the loop and stops the *master*, *slave*, and *tlm_fifo task*s.

```
task simulation is
      entry start;
      entry stop;
end simulation;
task body simulation is
begin
      accept start;
      loop
         select
         accept stop;
              exit;
         else
            master.start;
            slave.start;
         end select;

      end loop;
      master.stop;
      slave.stop;
    fifo_channel.tlm_fifo.stop;
end simulation;
```

Figure 10. Simulation task declaration and body

The process described above is an exact modeling of channels as expected in transaction level design. This description is more accordant to behavior of hardware than if the description were done with C++ and SystemC [1].

6. Implementation of a Network-on-Chip using SystemAda

In order to show the capabilities of SystemAda for modeling of digital systems at transaction level, we have implemented a Network-on-Chip (NoC) using SystemAda. The NoC architecture implemented in this paper is shown in Figure . This NoC has 7 switches and 6 processing elements. Switches are shown as squares, processors as circles, and channels between modules as straight lines. Each switch has a connected processing element except for switch4. Switches are connected to their adjacent switches by fifo channels which are named F1 to F8 respectively. In a similar way, each switch is connected to its corresponding processor by a fifo channel. Our NoC architecture uses a dynamic routing algorithm and works based on Store and Forward (SaF) packet switching method. The routing algorithm will be discussed later in this section. Processor1 and Prcoessor7 are the master processors of the NoC and input and output ports of the circuit are located at these processors respectively. Processor1 is in charge of reading data packets from an input file and sending them to Switch1. When Switch1 receives an incoming packet, sends it to appropriate output port that is decided by the routing algorithm. Upon arrival at destination, the packet is processed by corresponding processor and then, is sent toward Switch7. When Switch7 receives a packet, sends it to Processor7 which writes the packet data into some output file.

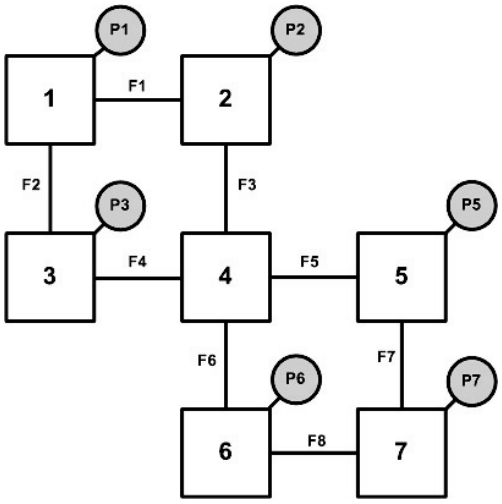


Figure 11. NoC architecture implemented using SystemAda

Figure 12 shows the body of the procedure which implements the functionality of our NoC. A task is defined for each processor and switch in the NoC. Each task has two entries, start and stop, except for Switch4 and switch7. Because Switch4 and Switch7 receive packets from two neighboring switches, they have separate entries for each input port instead of a single start entry. Swicth4 has two entries named start2 and start3 and Switch7 has entries start5 and strart6. Simulation task controls the simulation of the NoC design and has two entries for start and stop the simulation.

```
WITH
fifo1,fifo2,fifo3,fifo4,fifo5,fifo6,fifo7,fifo8,fifop1,fifop2,fifo3,fifo5,fifo6,fifop7;
PROCEDURE SystemAda_NoC IS
```

```
TASK processor1 IS
   ENTRY start;
   ENTRY stop;
END processor1;
...
TASK processor7 IS
   ENTRY start;
   ENTRY stop;
END processor7;

TASK switch1 IS
   ENTRY start;
   ENTRY stop;
END switch1;
...
TASK switch7 IS
   ENTRY start;
   ENTRY stop;
END switch7;

TASK simulation IS
   ENTRY start;
   ENTRY stop;
END simulation;
```

Figure 12. SystemAda_NoC procedure

Figure 13113 shows the body of processor1 task. The packets that will be sent to the network are written in a file named input_file. At the start of the task, input_file is opened. Then, the task enters an infinite loop in which the start entry is checked first. If the end of input_file is not reached, a packet is read from input_file and is added to fifop1. Fifop1 is the fifo between processor1 and switch1. Then, start entry of swicth1 is called and switch1 starts it operation. If stop has been called, the loop terminates.

```
TASK BODY processor1 IS
      input_file : FILE_TYPE;
BEGIN
      OPEN(input_file);
      LOOP
            SELECT
                  ACCEPT start DO
                        IF(not(END_OF_FILE(input_file))) THEN
                              GET(input_file,fifop1.input);
                              fifop1.tlm_fifo.add;
                              switch1.start;
                        END IF;
                  END start;
                  OR
                  ACCEPT stop;
                        EXIT;
            END SELECT;
      END LOOP;
      CLOSE(input_file);
END processor1;
```

Figure 131. Processor1 task body

Figure 14214 shows the body of switch1 task. A Boolean variable called flag is defined which is used in making routing decisions. At the beginning of the task, it enters an infinite loop. After accepting *start* entry, if the fifo between switch1 and processor1 (fifop1) is not empty, a packet is removed from it and is routed based on the routing algorithm that we describe here. If the switch has only one output port, the packet is sent to that port. Otherwise, two situations may arise. If the packet is targeted for an immediate neighbor of this switch, it will be sent to the corresponding port. Otherwise, the packet will be sent to one of the output ports alternatively using flag variable. As shown in Figure 14214, if the packet is targeted for Switch2 or if the packet is not targeted for switch3 and flag is false, the packet is sent to Switch2 and start entry for this switch is called. Otherwise, the packet is sent to Switch3 and start entry for this Switch is called. Upon calling stop entry of Switch1 task, the loop terminates.

```
TASK BODY switch1 IS
      flag : BOOLEAN := FALSE;
BEGIN
      LOOP
            SELECT
                  ACCEPT start DO
                        IF (fifop1.empty_flag=FALSE) THEN
                              fifop1.tlm_fifo.remove;
                              IF (fifop1.output.destination=2 OR
(fifop1.output.destination/=3 and flag=FALSE)) THEN
                                    flag:=TRUE;
                                    fifo1.input:=fifop1.output;
                                    fifo1.tlm_fifo.add;
                                    switch2.start;
                              ELSE
                                    flag:=FALSE;
                                    fifo2.input:=fifop1.output;
                                    fifo2.tlm_fifo.add;
                                    switch3.start;
                              END IF;
                        END IF;
                  END start;
                  OR
                  ACCEPT stop;
                        EXIT;
            END SELECT;
      END LOOP;
END switch1;
```

Figure 142. Switch1 task body

Figure 153 15 shows the body of simulation task. After accepting *start* entry, the task enters an infinite loop in which the stop entry is checked first. If stop has been called, the loop terminates. Otherwise, start entry of processor1 task is called and processor1 starts its simulation. Processor1 will call the start entry of switch1 task and switch1 will call the appropriate entry of switch2 or switch3 based on the routing decision. Upon arriving at destination, the packet will be processed by the destination processor and then routed toward Switch7 where it is written into the output file. This process repeats for the rest of the packets

generated at processor1. Once he stop entry of simulation task is called, the loop terminates and stop entries for switches, processor, and fifo tasks including fifos between switches and fifos between switches and their processors are called and simulation ends.

```
TASK BODY simulation IS
BEGIN
    ACCEPT start;
        LOOP
            SELECT
                ACCEPT stop;
                    EXIT;
                ELSE
                    processor1.start;
            END SELECT;
        END LOOP;
    switch1.stop;
    …
    switch7.stop;
    processor1.stop;
    …
    processor7.stop;
    --stop all fifo's and fifop's
    …
END simulation;
```

Figure 153. Simulation task body

7. Conclusion

Since digital designs are getting more complex, the need for a system description language that supports TLM is obvious. An important requirement at this level is to be able to run hardware of a system along with its software. Ada with intrinsic concurrency, early error detection, and extensive support for multithreading and multiprocessing would be a good choice to satisfy this requirement.

Other features of an HDL at a certain level of abstraction include link to its immediate lower abstraction level. In this paper, we showed how an RTL package could be used to override Ada with a register transfer capability.

Finally, an essential feature of an HDL at the transaction level is to be able to model transaction based communication channels for concurrent communication between processes. For this purpose, we showed how Ada could be used for defining the *tlm_fifo* channel functionality.

With the features inherent in Ada and packages added to this language, it could be a good candidate for transaction level description of hardware. Such a language is a required part of evolution of hardware design to higher levels of abstraction.

7. References

[1] S. Mirkhani, and Z.Navabi, *The VLSI Handbook*, Chapter 86, CRC Press, 2ed Edition, 2006.

[2] S. Wong, and Gertrude Levine, "Kernel Ada to Unify Hardware and Software Design", *Proc. Annual ACM SIGAda International Conference on Ada*, 1998, pp. 28-38.

[3] "Reusable Software Components for Reusable Hardware Designs", 2009-01-10, Available at: http://alpha.fdu.edu/~levine/reuse_course/columns/HDL_column.

[4] "A History of Object-Oriented Programming Languages and their Impact on Program Design and Software Development", 2009-01-10, Available at: http://jeffsutherland.com/papers/Rans/OOlanguages.pdf.

[5] R. Goering, "ESC: Ada 2005 Speaks to Real-time Embedded Applications", 2007-4-2, EE Times, Available at: http://www.embedded.com/news/embeddedindustry/198701828?_requestid=308128.

[6] J. E. Sammet, "Why Ada is not Just another Programming Language", *Communications of the ACM*, vol. 29, no. 8, August 1986, pp. 722-732.

[7] S. Swan, "OSCI SystemC TLM", 2009-01-10, Available at: http://www-ti.informatik.uni-tuebingen.de/~systemc/Documents/Presentation-13-OSCI_2_swan.pdf.

[8] "SystemC TLM1.0", 2009-01-10, Available at: http://www.systemc.org/home.

[9] Ada Reference Manual, ISO/IEC 8652:2007(E) Ed. 3, pp. 471-474.

[10] D. A. Wheeler, "Lovelace tutorial", Section 16.1-General Information on Interfacing to Other Languages, 2009-01-10, Available at: www.dwheeler.com/lovelace.

[11] GHDL guide, section 5.8.5 Linking with Ada, 2009-01-10, Available at: http://ghdl.free.fr/ghdl/Linking-with-Ada.html#Linking-with-Ada.

[12] M. Mills, and G Peterson, "Hardware/Software Codesign: VHDL and Ada 95 Code Migration and Integrated Analysis", *Proc. Annual ACM SigAda international conference on Ada*, 1998, pp. 18-27.

[13] Ada Reference Manual, ISO/IEC 8652:2007(E) Ed. 3, pp. 181-186.

[14] "Introductory Ada Concurrency Summary", 2009-01-10, Available at: http://www.seas.gwu.edu/~csci51/fall99/ada_task.html.

[15] D. A. Wheeler, "Lovelace tutorial", Section 13.2-Creating and Communicating with Tasks, 2009-01-10, Available at: http://www.dwheeler.com/lovelace/s13s2.htm.

[16] Negin Mahani, Parniyan Mokri, Zainalabedin navabi, "System Level Hardware Design and Simulatin with System Ada" ACM SIGADA AdaLetters, vol. XXIX , no. 1 , April 2009 , pp.19 -22.

[17] Negin Mahani, Parniyan Mokri, Mahshid Sedghi, Zainalabedin navabi, "System Ada : An Ada based Syste- Level Hardware Description Language" ACM SIGADA AdaLetters, vol. XXIX , no. 2 , August 2009 , pp. 15-19.

Gem #96: Code Archetypes for Real-Time Programming - Part 4

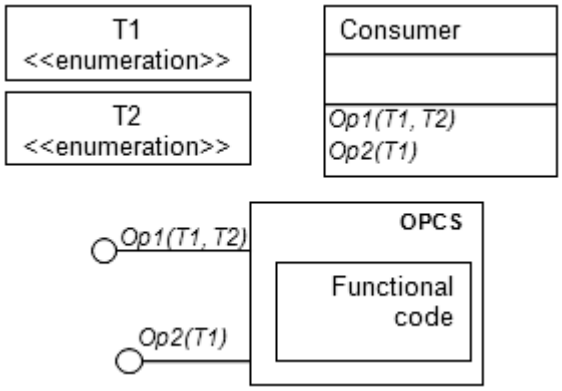
Author: Marco Panunzio, *University of Padua*

Let's get started…

In the previous Ada Gem we started to describe a complete archetype for a sporadic task. We illustrated the structure of the task and the realization of a complex queuing policy for its synchronization agent (OBCS). In this Ada Gem, we complete the picture with the description of the OPCS, which contains the functional code executed by the sporadic task, and show how we complete the declaration of the OBCS and of the provided interface exposed to clients of the sporadic task.

Sporadic Task -- Functional code and complete OBCS

It is now time to create the functional code for the procedures executed by our sporadic task. Suppose we want a Consumer that provides operations *Op1* and *Op2* as depicted in the figure below.



The operations *Op1* and *Op2* would be included in an OPCS structure which encapsulates their respective functional code, decoupling it from other nonfunctional concerns. The OPCS is then embedded in the sporadic task structure.

First, we declare two simple enumeration types, *T1* and *T2*, in a separate package, to be used by our functional code:

```
package Types is
   type T1 is (F1, F2);
   T1_Default_Value : constant T1 := F1;
   type T2 is (X1, X2);
   T2_Default_Value : constant T2 := X1;
end Types;
```

Then in another package we declare a new type, say *Consumer_FC*, that extends Controlled (thus it is a tagged type) and has two primitive procedures *Op1(T1, T2)* and *Op2(T1)*:

```
with Types;
with Ada.Finalization; use Ada.Finalization;

package Consumer is

   type Consumer_FC is new Controlled with private;
   type Consumer_FC_Ref is access all Consumer_FC'Class;
   type Consumer_FC_Static_Ref is access all Consumer_FC;
   type Consumer_FC_Arr is array(Standard.Integer range <>) of
Consumer_FC_Ref;
   type Consumer_FC_Arr_Ref is access Consumer_FC_Arr;

   overriding
   procedure Initialize(This : in out Consumer_FC);

   procedure Op1 (This : in out Consumer_FC; a : in Types.T1; b : in
Types.T2);
   procedure Op2 (This : in out Consumer_FC; a : in Types.T1);

private

    type Consumer_FC is new Controlled with null record ...;

end Consumer;
```

Consumer_FC is the type that represents what we called the OPCS. The sequential code is given in the bodies of the two procedures *Op1* and *Op2*:

```
package body Consumer is

   -- procedure Initialize omitted

   procedure Op1(This : in out Consumer_FC; a : in Types.T1; b : in
Types.T2) is
   begin
   -- User-defined sequential code here --
   end Op1;

   procedure Op2 (This : in out Consumer_FC; a : in Types.T1) is
   begin
   -- User-defined sequential code here --
   end Op2;

end Consumer;
```

Now let's complete the definition of the OBCS and discuss the instantiation of the sporadic task.

```
with System;
with Types;
with System_Types;
with Ada.Real_Time;

with Consumer;
with Ada.Real_Time; use Ada.Real_Time;
```

```
package Op1_Op2_Sporadic_Consumer is

   use System; use Types;

   use System_Types;

   -- Nested generic package for instantiating a sporadic task:

   generic
      Thread_Priority : Priority;
      Ceiling : Priority;
      MIAT : Integer;
      -- The OPCS instance
      OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
   package My_Sporadic_Factory is

      procedure Op1(a : in T1; b : in T2);
      procedure Op2(a : in T1);

   private
      -- ...
   end My_Sporadic_Factory;

private

    Param_Queue_Size : constant Integer := 3;
    OBCS_Queue_Size : constant Integer := Param_Queue_Size * 2;

    -- Create data structures to reify invocations of Op1

    type Op1_Param_T is new Param_Type with record
       OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
       a : T1;
       b : T2;
    end record;

    type Op1_Param_T_Ref is access all Op1_Param_T;

    type Op1_Param_Arr is array(Integer range <>) of aliased
Op1_Param_T;

    overriding
    procedure My_OPCS(Self : in out Op1_Param_T);

    -- Create data structures to reify invocations of Op2

     type Op2_Param_T is new Param_Type with record
       OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
       a : T1;
    end record;

    type Op2_Param_T_Ref is access all Op2_Param_T;

    type Op2_Param_Arr is array(Integer range <>) of aliased
Op2_Param_T;
```

```
    overriding
    procedure My_OPCS(Self : in out Op2_Param_T);

    -- Create an OBCS that matches the interface of the OPCS (FC)
    protected type OBCS
      (Ceiling : Priority;
       Op1_Params_Arr_Ref_P : Param_Arr_Ref;
       Op2_Params_Arr_Ref_P : Param_Arr_Ref)
    is
      pragma Priority(Ceiling);

      entry Get_Request(Req : out Request_Descriptor_T; Release : out
Time);
      procedure Op2(a : in T1);
      procedure Op1(a : in T1; b : in T2);

    private

       -- The queue system for the OBCS
       OBCS_Queue : Sporadic_OBCS(OBCS_Queue_Size);
       -- Arrays to store a set of reified invocations for Op1 and Op2
       Op1_Params : Param_Buffer_T(Param_Queue_Size) :=
          (Size => Param_Queue_Size, Index => 1, Buffer =>
Op1_Params_Arr_Ref_P.all);
       Op2_Params : Param_Buffer_T(Param_Queue_Size) :=
          (Size => Param_Queue_Size, Index => 1, Buffer =>
Op2_Params_Arr_Ref_P.all);
       Pending : Standard.Boolean := False;

    end OBCS;

end Op1_Op2_Sporadic_Consumer;
```

In essence, in the specification above: (i) we declare a nested generic package (*My_Sporadic_Factory*) that we use to instantiate a sporadic task. In this manner we can instantiate several sporadic tasks which only differ in their timing attributes and properties (MIAT, priority and ceiling priority for the OBCS); the generic package provides an interface to the rest of the system that matches its OPCS (it provides *Op1* and *Op2* in our case); (ii) in the private part of the parent package (*Op1_Op2_Sporadic_Consumer*), we create the data structures to store reified invocations of *Op1* and *Op2*. This is done by extending *Param_Type* (defined in System_Types, see the previous Ada Gem in this series) by new types *Op1_Param_T* and *Op2_Param_T* that are records containing the parameters of the call and a reference to the OPCS (an access to *FC* in our case). Additionally, we override procedure *My_OPCS*. Therefore, when *My_OPCS* is called on *Op1_Param_T* or *Op2_Param_T*, it will dispatch to the appropriate procedure that we later define in the body of this package. The reader can check again that this is what really happens when the sporadic task type (defined in the previous Gem in this series) calls the procedure *My_OPCS* after fetching the request descriptor from the OBCS.

```
with Ada.Real_Time; use Ada.Real_Time;

with Sporadic_Task;
```

```
with Types; use Types;

package body Op1_Op2_Sporadic_Consumer is

   use System_Types;

   -- Redefinition of My_OPCS. Call Consumer_FC.Op1 and set In_Use to
False.

   procedure My_OPCS(Self : in out Op1_Param_T) is
   begin
      Self.OPCS_Instance.Op1(Self.a, Self.b);
      Self.In_Use := False;
   end My_OPCS;

   -- Redefinition of My_OPCS. Call Consumer_FC.Op2 and set In_Use to
False.
   procedure My_OPCS(Self : in out Op2_Param_T) is
   begin
      Self.OPCS_Instance.Op2(Self.a);
      Self.In_Use := False;
   end My_OPCS;

   protected body OBCS is

      procedure Update_Barrier is
      begin
         Pending := (OBCS_Queue.Pending) > 0;
      end Update_Barrier;

      -- Get_Request stores the time of the release of the task,
      -- gets the next request (according to the OBCS queuing policy),
      -- and updates the guard.

      entry Get_Request (Req : out Request_Descriptor_T; Release : out
Time) when Pending is
      begin
         Release := Clock;
         Get(OBCS_Queue, Req);
         Update_Barrier;
      end Get_Request;

      -- When a client calls Op1, the request is reified and put in the
OBCS queue.

      procedure Op1(a : in T1; b : in T2) is
      begin
        if Op1_Params.Buffer(Op1_Params.Index).In_Use then
           Increase_Index(Op1_Params);
        end if;

        Op1_Param_T_Ref(Op1_Params.Buffer(Op1_Params.Index)).a := a;
        Op1_Param_T_Ref(Op1_Params.Buffer(Op1_Params.Index)).b := b;
        Put(OBCS_Queue, START_REQ,
Op1_Params.Buffer(Op1_Params.Index));
        Increase_Index(Op1_Params);
        Update_Barrier;
```

```
      end Op1;

      -- When a client calls Op2, the request is reified and put in the
OBCS queue.

      procedure Op2(a : in T1) is
      begin
         if Op2_Params.Buffer(Op2_Params.Index).In_Use then
             Increase_Index(Op2_Params);
         end if;

         Op2_Param_T_Ref(Op2_Params.Buffer(Op2_Params.Index)).a := a;
         Put(OBCS_Queue, ATC_REQ, Op2_Params.Buffer(Op2_Params.Index));
         Increase_Index(Op2_Params);
         Update_Barrier;
      end Op2;

   end OBCS;

   package body My_Sporadic_Factory is

       Op1_Par_Arr : Op1_Param_Arr(1..Param_Queue_Size) := (others =>
                     (False,
                      OPCS_Instance,
                      T1_Default_Value,
                      T2_Default_Value));

       Op1_Ref_Par_Arr : aliased Param_Arr := (Op1_Par_Arr(1)'access,
           Op1_Par_Arr(2)'access, Op1_Par_Arr(3)'access);

       Op2_Par_Arr : Op2_Param_Arr(1..Param_Queue_Size) := (others =>
                    (false,
                     OPCS_Instance,
                     T1_Default_Value));

       Op2_Ref_Par_Arr : aliased Param_Arr := (Op2_Par_Arr(1)'access,
            Op2_Par_Arr(2)'access, Op2_Par_Arr(3)'access);

       -- Creation of the OBCS
       Protocol : aliased OBCS(Ceiling, Op1_Ref_Par_Arr'access,
                                        Op2_Ref_Par_Arr'access);

       -- Indirection to Get_Request of the OBCS

       procedure Getter(Req : out Request_Descriptor_T; Release : out
Time) is
       begin
          Protocol.Get_Request(Req, Release);
       end Getter;

       -- Instantiate the generic package using the procedure above

       package My_Sporadic_Task is new Sporadic_Task(Getter);

       Thread : My_Sporadic_Task.Thread_T(Thread_Priority, MIAT);

       -- When a client calls Op1, redirect the call to the OBCS
```

```
         procedure Op1(a : in T1; b : in T2) is
         begin
            Protocol.Op1(a, b);
         end Op1;

         -- When a client calls Op2, redirect the call to the OBCS
         procedure Op2(a : in T1) is
         begin
            Protocol.Op2(a);
         end Op2;

   end My_Sporadic_Factory;

end Op1_Op2_Sporadic_Consumer;
```

The package body above overrides *My_OPCS* for each operation provided to external clients (*Op1* and *Op2*). The overriding simply ensures that *My_OPCS* calls the correct operation with the stored parameter of the original request and then signals that the parameters are no longer in use (which ensures correct management of the circular buffers in the OBCS).

The body of the OBCS follows the same logic as the simpler OBCS described in Gem #92. Procedure *Op1* and *Op2* are simply extended to reify call requests and correctly store the parameters of the calls in the request descriptor.

Finally, in the body of the generic package *My_Sporadic_Factory*, we create an OBCS with a defined ceiling priority and the queues to store the parameters of reified calls to *Op1* and *Op2*. The sporadic thread is instantiated in the same package, and we complete the picture by redirecting the calls from *Op1* and *Op2*, in the provided interface of the task structure, to the operations with the same names in the OBCS.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #97: Reference Counting in Ada - Part 1

Author: Emmanuel Briot, *AdaCore*

Let's get started…

Memory management is typically a complex issue to address when creating an application, and even more so when creating a library to be reused by third-party applications. It is necessary to document which part of the code allocates memory and which part is supposed to free that memory. As we have seen in a previous Gem, a number of tools exist for detecting memory leaks (gnatmem, GNATCOLL.Memory or valgrind). But of course, it would be more convenient if the memory were automatically managed.

Some languages include an automatic garbage collector. The Ada Reference Manual has an implementation permission allowing a conformant compiler to provide one, although none of the mainstream compilers do so. Ada's design allows implementations to use the stack in many situations where other languages use the heap; this reduces the need for a garbage collector.

An alternative implementation for getting automatic memory management is to use reference counting: every time some object is allocated, a counter is associated with it. This counter records how many references to that object exist. When that counter goes down to zero, it means the object is no longer referenced in the application and can therefore be safely deallocated.

The rest of this Gem will show how to implement such a mechanism in Ada. As we will see, there are a number of minor but delicate issues involved, so implementing such types is not as trivial as it first seems. The GNAT Components Collection (GNATcoll) now includes a reusable generic package that simplifies this, and we will discuss this briefly at the end of this Gem.

As stated above, we need to associate a counter with the objects of all types we want to monitor. The simplest is to create a tagged type hierarchy where the root type defines the counter:

```
   type Refcounted is abstract tagged private;
   procedure Free (Self : in out Refcounted) is null;

private

   type Refcounted is abstract tagged record
      Refcount : Integer := 0;
   end record;
```

This approach is mostly suitable when building a reusable library for reference-counted types, such as GNATcoll. If you just want to do this once or twice in your application,

you can simply add a new Refcount field to your record type (which doesn't need to be tagged).

Next, we need to determine when to increment and decrement this counter. In some languages this counter needs to be manually modified by the application whenever a new reference is created, or when one is destroyed. This is, for instance, how the Python interpreter is written (in C). But we can do better in Ada, by taking advantage of controlled types. The compiler calls special primitive operations each time a value of such a type is created, copied, or destroyed.

If we wrap a component of a simple access type in a type derived from Ada.Finalization.Controlled, we can then have the compiler automatically increment or decrement the reference count of the designated entity each time a reference is established or removed. We thus create a smart pointer: a pointer that manages the life cycle of the block of memory it points to.

```
   type Refcounted_Access is access all Refcounted'Class;
   type Ref is tagged private;

   procedure Set (Self : in out Ref; Data : Refcounted'Class);
   function Get (Self : Ref) return Refcounted_Access;
   procedure Finalize (P : in out Ref);
   procedure Adjust   (P : in out Ref);

private

   type Ref is new Ada.Finalization.Controlled with record
       Data : Refcounted_Access;
   end record;
```

Let's first see how a user would use the type. Note that Get returns an access to the data. This might be dangerous, since the caller might want to free the data (which should remain under control of Ref). In practice, the gain in efficiency is worth it, since it avoids making a copy of a Refcounted'Class object. This is also essential if we want to allow the user to easily modify the designated entity. The user is ultimately responsible for ensuring that the lifetime of the returned value is compatible with the lifetime of the corresponding smart pointer.

```
declare
  type My_Data is new Refcounted with record
     Field1 : ...;
  end record;

  R1 : Ref;

begin
  Set (R1, My_Data'(Refcounted with Field1 => ...));
  --  R1 holds a reference to the data

  declare
```

```
      R2 : Ref;
   begin
      R2 := R1;
      --  R2 also holds a reference to the data (thus 2 references)

      ...
      --  We now exit the block. R2 is finalized, thus only 1 ref left

   end;

   Put_Line (Get (R1).Field1);  --  For instance

   --  We now leave R1's scope, thus refcount is 0, and the data is
freed.
end;
```

Now let's look at the details of the implementation. First consider the two subprograms for setting and getting the designated entity. Note that the default value for the reference count is zero in the Refcounted type. The implementation of Set is slightly tricky: it needs to decrement the reference count of the previously designated entity, and increment the reference count for the new data. Instead of calling Adjust and Finalize explicitly (which is not a recommended practice when it can be avoided), we use an aggregate and let the compiler generate the calls for us.

```
procedure Set (Self : in out Ref; Data : Refcounted'Class) is
   D : constant Refcounted_Access := new Refcounted'Class'(Data);
begin
   if Self.Data /= null then
      Finalize (Self); -- decrement old reference count
   end if;

   Self.Data := D;
   Adjust (Self);  -- increment reference count (set to 1)
end Set;

function Get (P : Ref) return Refcounted_Access is
begin
    return P.Data;
end Get;
```

In GNATCOLL.Refcount, we provide a version of Set that receives an existing access to Refcount'Class, and takes responsibility for freeing it when it is no longer needed. The implementation is very similar to the above (although we need to be careful that we do not Finalize the old data if it happens to be the same as the new, since otherwise we might end up freeing the memory).

Adjust is called every time a new reference is created. Nothing special here:

```
overriding procedure Adjust (P : in out Ref) is
begin
    if P.Data /= null then
       P.Data.Refcount := P.Data.Refcount + 1;
    end if;
```

```
end Adjust;
```

The implementation of Finalize is slightly more complicated: the Ada reference manual indicates that a Finalize procedure should always be idempotent. An Ada compiler is free to call Finalize multiple times on the same object, in particular when exceptions occur. This means we must be careful not to decrement the reference counter every time Finalize is called, since a given object only owns one reference. Hence the following implementation:

```
overriding procedure Finalize (P : in out Ref) is
   Data : Refcounted_Access := P.Data;
begin
   --  Idempotence: the next call to Finalize will have no effect
   P.Data := null;

   if Data /= null then
      Data.Refcount := Data.Refcount - 1;
      if Data.Refcount = 0 then
         Free (Data.all);  --  Call to user-defined primitive
         Unchecked_Free (Data);
      end if;
   end if;

end Finalize;
```

That's it for the basic implementation. The next Gem in this series will discuss issues of task safety associated with reference-counted types.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #98: High Performance Multi-core Programming - Part 2

Author: Pat Rogers, *AdaCore*

Let's get started…

Chameneos-Redux is part of the Computer Language Shootout, a suite of benchmarks that compares the implementations of various programming languages across different kinds of applications and platforms. The program is required to perform a specified number of rendezvous between mythical "chameneos" creatures, where each creature is represented by a distinct thread. Each rendezvous is symmetric, in that the participating creatures can be either the caller or the called member in any given encounter. The multi-core benchmark results for all implementations of Chameneos-Redux are available here: Chameneos-Redux.

In the previous Gem in this series, we made the point that design trumps tuning: when it comes to performance, no amount of tweaking can compensate for an inherently slow design. In this Gem we explore another important aspect of the design: minimizing resource conflicts via processor assignments for threads. This issue arises because the Chameneos-Redux benchmark requires two distinct "games," in which a number of threads perform the required number of rendezvous. (One game has a different number of threads, but both must execute the same number of rendezvous.) When multiple processors are available the program runs the two games concurrently, so it would be possible for the two sets of threads to run on the same cores.

To prevent the threads of one game from interfering with the threads of another game running concurrently, we permanently assign threads to processors rather than let the operating system assign them dynamically. Specifically, we allow a thread to execute on any of the cores within a single assigned processor. All of the leading Chameneos-Redux implementations use the same approach to prevent this sharing.

Assigning threads to processors is specified in terms of "slots" instead of directly in terms of processors. A slot is an integer value corresponding to a processor number, but since there are likely more threads than processors, when the slot number exceeds the number of processors present in the machine, the slot number "wraps around" back to the beginning processor. As a result, there is effectively no limit to the number of slots available. This does not prevent threads from sharing (cores on) processors, it simply makes assignment convenient. If there are too many threads, sharing will be unavoidable.

Slot assignment is achieved using pragma Task_Info within the task type declaration representing the chameneos creatures' threads. The argument to the pragma is an access value designating a value of type Thread_Attributes, which on the benchmark operating system is a record type containing a single affinity bit-mask component. The affinity mask indicates the cores on which tasks may execute. We define a function Affinity that returns a value of that type. The following code fragment illustrates use of the pragma and the function:

```
 task type Thread (This : access Creature;  Slot : Natural) is
   pragma Task_Info (new Thread_Attributes'(CPU_Affinity => Affinity
(Slot)));
 end Thread;
```

The argument to the Affinity function is a task type discriminant indicating the slot to use. A slot logically contains an affinity mask that indicates the cores in the corresponding processor, and it is this mask that is returned by the function.

Slot zero is used to hold a bit-mask indicating all the cores known to the system. We import function Sched_Getaffinity to get this value. For example, imagine the function returns a mask with the first eight bits enabled, indicating that a total of eight cores are available. Slot zero will then hold a bit mask with eight bits set. Assuming two cores per processor, and assuming that every two contiguous bits represent cores on the same processor, the following illustrates the resulting masks per slot:

```
            Bit #
Slot #   0123456789...
    0    1111111100
    1    1100000000
    2    0011000000
    3    0000110000
    4    0000001100
    5    1100000000
    6    0011000000
```

Observe that at slot 5 the two cores on the first processor are again involved. You can see the details of the implementation by examining the Chameneos.Processors package.

Getting back to the game, the main program determines whether the target is a multi-core machine and assigns slots for the two required games, and thus the threads in the games, accordingly:

```
   if Processor_Count < 4 then  -- run the games sequentially
      Game1.Start (Game1_Creature_Colors, N, Slot => 0);
      Game1.Await_Completion;
      Game2.Start (Game2_Creature_Colors, N, Slot => 0);
      Game2.Await_Completion;
   else -- run the games in parallel
      Game1.Start (Game1_Creature_Colors, N, Slot => 1);
      Game2.Start (Game2_Creature_Colors, N, Slot => 2);
      Game1.Await_Completion;
      Game2.Await_Completion;
   end if;
```

In this manner the two games do not share processors, so they do not conflict with each other.

In the next Gem in this series we will explore another aspect of the implementation relative to the cache, specifically a user-defined storage allocator that allocates dynamic memory on cache-aligned boundaries.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #99: Reference Counting in Ada - Part 2: Task Safety

Author: Emmanuel Briot, *AdaCore*

Let's get started…

In Part 1, we described a reference-counted type that automatically frees memory when the last reference to it disappears. But this type is not task safe: when we decrement the counter, it might happen that two tasks see it as 0, and thus both will try to free the data. Likewise, the increment of the counter in Adjust is not an atomic operation, so it is possible that we will be missing some references.

In some applications this restriction is not a big issue (for instance, if there are no tasks, or if the types are only ever used from a single task). However, let's try to improve the situation.

The traditional solution is to use a lock while we are manipulating the counter. We could, for instance, use a protected type for this. However, this means that a nontasking application using our reference-counted types would have to initialize the whole tasking run-time, which could impact execution somewhat, since part of the code goes through slower code paths.

GNAT provides a global lock that we can reuse for that, and that does not require the full tasking run-time. We could use that lock in a function that changes the value of the counter atomically. We need to return the new value from that function: changing the value atomically solves the problem we highlighted for Adjust, but not the one we showed for Finalize, where two tasks could see the value as 0 if they read it separately.

```
function Atomic_Add
  (Ptr : access Integer; Inc : Integer) return Integer
is
   Result : Integer;
begin
   GNAT.Task_Lock.Lock;
   Ptr.all := Ptr.all + Value;
   Result := Ptr.all;
   GNAT.Task_Lock.Unlock;
   return Result;
end Atomic_Add;
```

On some systems there is actually a more efficient way to do this, by using an intrinsic function: this is a function provided by the compiler, generally implemented directly in assembly language using low-level capabilities of the target machine. We need special handling to check whether this facility is available, but if it is, we no longer need a lock. The GNATCOLL.Refcount package takes full advantage of this.

```
function Atomic_Add
  (Ptr : access Integer; Inc : Integer) return Integer
is
```

```
   function Intrinsic_Sync_Add_And_Fetch
     (Ptr   : access Interfaces.Integer_32;
      Value : Interfaces.Integer_32) return Interfaces.Integer_32;
   pragma Import
     (Intrinsic, Intrinsic_Sync_Add_And_Fetch,
"__sync_add_and_fetch_4");
begin
   return Intrinsic_Sync_Add_And_Fetch (Ptr, Value);
end Atomic_Add;
```

(Note: In actual practice, it would be necessary to declare the access parameter of function Atomic_Add with type Interfaces.Integer_32, for type compatibility with the intrinsic.)

Once we have this Atomic_Add function we need to modify our reference-counted type implementation. The first change is to declare the Refcount field as aliased, in the definition of Refcounted. We then revise the code as follows:

```
overriding procedure Adjust (P : in out Ref) is
   Dummy : Integer;
begin
   if P.Data /= null then
      Dummy := Atomic_Add (P.Data.Refcount'Access, 1);
   end if;
end Adjust;

overriding procedure Finalize (P : in out Ref) is
   Data : Refcounted_Access := P.Data;
begin
   P.Data := null;
   if Data /= null
      and then Atomic_Add (Data.Refcount'Access, -1) = 0
   then
      Free (Data.all);
      Unchecked_Free (Data);
   end if;
end Finalize;
```

The last Gem in this series will talk about a different kind of reference, generally known as a weak reference.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #100: Reference Counting in Ada - Part 3: Weak References

Author: Emmanuel Briot, *AdaCore*

Let's get started…

As we mentioned in the first two parts of this Gem series, GNATCOLL now includes a package that provides support for memory management using reference counting, including taking advantage of the efficient synchronized add-and-fetch intrinsic function on systems where it is available.

There is one thing that reference-counted types cannot handle as well as a full-scale garbage collector: cycles. If A references B which references A, neither of them will ever get freed. A garbage collector is often able to detect such cycles and deallocate all the objects as appropriate, but such a case cannot be handled automatically through reference counting. However, there's a variant approach that can handle such cases with only minor changes in the code.

Let's take an example: you are retrieving values from some container (a database for instance), and want to have a local cache to speed things up. The code would likely be organized as follows:

- Get a reference-counted value from the container. Its counter is 1.
- Put it in the cache for later use. The counter is now 2, since the cache itself owns a reference.
- When you are done using the value in your algorithm, you release the reference you had. Its counter goes down to 1 (the cache still owns the reference).

Because of the cache, the value is never freed from memory. This is not good, since memory usage will only keep increasing.

GNATCOLL provides a solution for this issue, through the use of *weak references*. This is a standard industry term for a special kind of reference: you have a type that points to the same object as a true reference-counted type would, but that type does not hold a reference. Thus, it does not prevent the counter from reaching 0, and the object from being freed.

When the deallocation occurs, the internal data of the weak reference is reset. Thus, if you retrieve the data stored in the weak reference, you get null, not an erroneous access to some freed memory (which might sooner or later result in a Storage_Error).

If we set up the cache so that it uses weak references, the code becomes:

- Get a reference-counted value from the container. Its counter is 1.
- Put it in the cache, through a weak reference. The counter is still 1.

- When you are done using the value, the counter goes down to 0, and the memory is freed.
- At this point, the cache still contains the weak reference, but the latter uses just a little memory.

Using slightly more complex code, it is possible, in fact, to remove the entry for the cache altogether when the value is freed, thus really releasing all memory to the system. Though GNATCOLL does provide a capability for using weak references, a future package will provide easier handling of such caches.

One way to implement weak references is by adding an extra pointer in type Refcount. GNATCOLL chooses to make this optional: if you want to systematically have that extra pointer in your data structure, you can use weak references. Otherwise, you still have access to the code we described in the first part of this Gem series.

We will not go into the details of the implementation for a weak reference. Interested parties can look at the code in GNATCOLL.Refcount.Weakref, which is relatively small.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #101: SOAP/WSDL server part

Author: Pascal Obry, *EDF R&D*

Let's get started…

This is the first part of a two-part Gem on SOAP (Simple Object Access Protocol).

In this Gem we will be building a SOAP server and you'll see that with Ada it is quite simple!

Let's take a simple package spec such as the following:

```
package Temperatures is

   type Celsius is new Float;
   type Fahrenheit is new Float;

   function To_Fahrenheit (C : Celsius) return Fahrenheit;
   function To_Celsius    (F : Fahrenheit) return Celsius;

end Temperatures;
```

The body is not shown here but it's part of the source packages that can be downloaded (see below).

The first step is to generate the WSDL (Web Service Description Language). A WSDL is an XML language for describing Web services. In the WSDL we find a description of the types and the specs of the routines. A WSDL is similar to an IDL but based on XML.

To generate the WSDL, AWS come with the ASIS-based **ada2wsdl** tool:

```
$ ada2wsdl temperatures.ads -a http://localhost:8888 -o
temperatures.wsdl
```

The options are:

```
-a http://...          Specifies the end-point for the Web services.

-o temperatures.wsdl   Outputs WSDL into temperatures.wsdl.
```

Out of this WSDL it's possible to generate stubs (for calling Web services) or skeletons (for implementing Web services). In this first part we're building a server, so we don't need the stubs. AWS comes with a second tool called **wsdl2aws** to generate all the necessary the code:

```
$ wsdl2aws -nostub -cb -spec temperatures -main soap_server
temperatures.wsdl
```

The options are:

```
-spec temperatures   To use the routines as implemented in Temperatures
                     unit.

-cb                  Generates the SOAP callbacks using the routines
                     found in the spec specified above.

-main soap_server    Generates a main named soap_server, this main
                     program starts the SOAP server by referencing a
SOAP
                     dispatcher using the callback routines.
```

Using the three options above is very handy for building a server that provides Web services and nothing more. The last actions are just to compile the server and run it:

```
$ gnatmake -gnat05 -Psoap_server
$ ./server
```

At this point the services are available on the network and can be called by other programs, possibly built with other languages (Java and C# are the most common ones).

In the second part of this series we will see how to call those services from Ada using AWS.

1.1.1.1 Attached Files

- soap_server.zip - (890 B)

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #102: SOAP/WSDL client part

Author: Pascal Obry, *EDF R&D*

Let's get started…

This is the second part of a two-part Gem series on SOAP and WSDL.

In this Gem we will be using a Web Service as described in a WSDL document. These services could be implemented in Java, C#, or Ada, because the WSDL is universal in the Web Services world.

In the previous Gem we generated a WSDL from a simple Ada spec. Let's use it to generate the necessary code to use these Web services. We again use the **wsdl2aws** tool, but this time to generate only the stubs:

```
$ wsdl2aws -f -noskel temperatures.wsdl
```

A set of packages is generated. Two are of interest to us at the moment, namely:

Package temperatures_service-types.ads, containing the types used by the Web services.

Package temperatures_service-client.ads, containing the Web services client spec.

For each Web Service routine, two specs are generated:

```
function To_Fahrenheit
  (C        : Celsius_Type;
   Endpoint : String := Temperatures_Service.URL;
   Timeouts : AWS.Client.Timeouts_Values :=
Temperatures_Service.Timeouts)
   return To_Fahrenheit_Result;

function To_Fahrenheit
  (Connection : AWS.Client.HTTP_Connection;
   C          : Celsius_Type)
   return To_Fahrenheit_Result;

--  Raises SOAP.SOAP_Error if the operation fails
```

The first connects and closes the connection for each call, whereas the second uses a persistent connection. The usage is straightforward. Now, let's build a small program which converts Celsius to Fahrenheit:

```
with Ada.Text_IO;
with Temperatures_Service.Client;
with Temperatures_Service.Types;

procedure SOAP_Client is
   use Ada;
```

```
   use Temperatures_Service;
   C : constant Types.Celsius_Type := 20.0;
   F : constant Types.Fahrenheit_Type := Client.To_Fahrenheit (C);
   package C_IO is new Text_IO.Float_IO (Types.Celsius_Type);
   package F_IO is new Text_IO.Float_IO (Types.Fahrenheit_Type);

begin
   Text_IO.Put ("Celsius    "); C_IO.Put (C, Aft => 1, Exp => 0);
   Text_IO.New_Line;
   Text_IO.Put ("Fahrenheit "); F_IO.Put (F, Aft => 1, Exp => 0);
   Text_IO.New_Line;
end SOAP_Client;
```

We can use the following simple project file to build this program:

```
with "aws";
project SOAP_Client is
   for Source_Dirs use (".");
   for Main use ("soap_client.adb");
end SOAP_Client;
$ gnatmake -gnat05 -Psoap_client
```

Now let's test it, first by starting the server we have built last week:

```
$ ./soap_server
```

Then running soap_client:

```
$ ./soap_client
Celsius    20.0
Fahrenheit 68.0
```

That's all there is to it. As we've shown, it's easy to use a Web Service in Ada when the WSDL is provided. It's still possible to use a Web Service without a WSDL, but in that case it would be necessary to hand-code it.

1.1.1.2 Attached Files

- soap_client.zip - (691 B)

Related Source Code

Gem #103: Code Archetypes for Real-Time Programming—Part 5

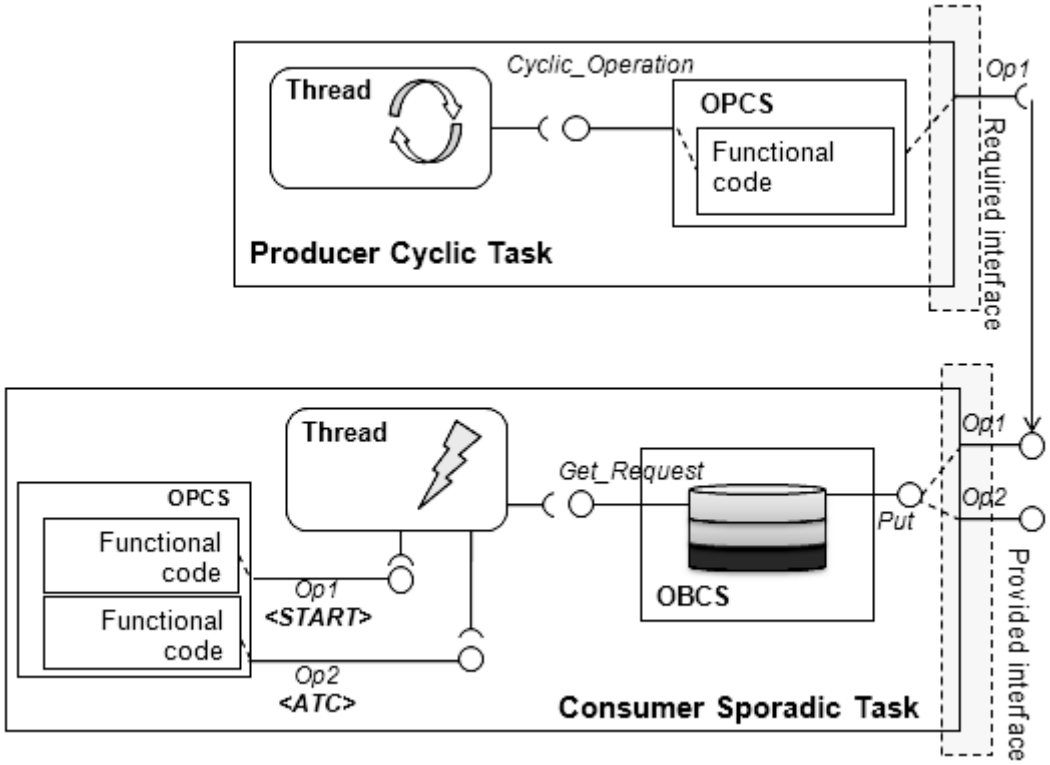
Author: Marco Panunzio, *University of Padua*

Let's get started…

In the previous Ada Gem we completed the creation of a complete sporadic task. In this Ada Gem that ends our mini-series, we want to complete the example by adding the realization of the communication between different tasks. In particular, we investigate how we can correctly manage the calls of operations outside a task. Those calls are performed from an OPCS and they have to be correctly routed to the endpoint of the communication.

Intertask communication -- A producer-consumer example

Suppose we want to realize the simple producer-consumer collaboration pattern depicted in the figure below.



The *Producer* is a cyclic task which, after some processing, produces some data that is sent to a *Consumer* sporadic task. Data is passed as a parameter of operation *Op1*.

This implies that we have to equip the *Producer* cyclic task with the means to communicate with the *Consumer* sporadic task. However, the task structure that we created encapsulates the functional code inside a structure called the OPCS. Therefore, inside the functional code of the Producer we cannot directly call Op1 of the Consumer

(i.e., the functional/sequential code), but we have to call the appropriate provided interface of the whole Consumer task.

Let us see how we can achieve this goal when creating the package Producer (analogous to the package Consumer that we presented in the previous Ada Gem of the series).

```
package Producer is

   type Producer_FC is new Controlled with private;
   type Producer_FC_Ref is access all Producer_FC'Class;
   type Producer_FC_Static_Ref is access all Producer_FC;
   -- [code omitted]

   overriding
   procedure Initialize(This : in out Producer_FC);

   procedure Op0 (This : in out Producer_FC);
   procedure Set_x(This : in out Producer_FC;
                   v : in Consumer.Consumer_FC_Ref);

private

   type Producer_FC is new Controlled with record
      x : Consumer.Consumer_FC_Ref;
   end record;

end Producer;
```

We are adding a member *x* in the record of *Producer_FC* that represents a reference to the Consumer that provides *Op1*, which consumes the data produced by the Producer. The reader should note that the static type of this reference is the OPCS of the Consumer (which was called *Consumer_FC*).

```
package body Producer is

   -- [procedure Initialize omitted]

   procedure Op0(This : in out Producer_FC) is
   begin
      This.x.Op1([T1_VALUE]);
   end Op1;

   procedure Set_x(This : in out Producer_FC;
                   v : in Consumer.Consumer_FC_Ref)
   is
   begin
      This.x := v;
   end Set_x;
end Producer;
```

In the body of procedure Op0 (where the sequential code executed by the Producer task is specified), we insert the call to *Op1* that is performed using reference *x*.

```
-- Package spec
type s1_T is new Consumer.Consumer_FC with record
   Op1_Ref : access procedure (a : in Types.T1; b : in Types.T2);
   Op2_Ref : access procedure (a : in Types.T1);
end record;

overriding
procedure Op1(This : in out s1_T; a : in Types.T1; b : in Types.T2);

overriding
procedure Op2(This : in out s1_T; a : in Types.T1);

-- Package body
procedure Op1(This : in out s1_T; a : in Types.T1; b : in Types.T2) is
begin
   This.Op1_Ref.all(a,b);
end Op1;
```

Above, in another package, we create a new type *s1_T*. This type extends *Consumer_FC* and adds a pointer to a procedure with the signature of *Op1*, the operation that we call at the Producer side, and *Op2*. We also override *Op1* for *s1_T*, so that a call to *Op1* is reissued to the pointer just defined.

Analogously, suppose we create a new type *s0_T* that extends *Producer_FC*.

In the few remaining code excerpts below, we complete the example with the instantiation of the cyclic task for the Producer and the sporadic task for the Consumer.

```
s0_Instance : aliased s0_T;
package My_Cyclic_Producer_Task is new
     Op0_Cyclic_Producer.My_Sporadic_Factory(
           Thread_Priority => 1,
           Period => 2000,
           OPCS_Instance =>
               Producer.Producer_FC(s0_Instance)'access);

s1_Instance : aliased s1_T;

package My_Sporadic_Consumer_Task is new
     Op1_Op2_Sporadic_Consumer.My_Sporadic_Factory(
           Thread_Priority => 2,
           Ceiling => 2,
           MIAT => 500,
           OPCS_Instance =>
               Consumer.Consumer_FC(s1_Instance)'access);
```

Note that we pass a pointer to *s0_Instance* (respectively *s1_Instance*) as the OPCS during the instantiation of the Producer (respectively Consumer) cyclic (respectively sporadic) task.

```
s1_Instance.Op1_Ref := My_Sporadic_Consumer_Task.Op1'access;
s1_Instance.Op2_Ref := My_Sporadic_Consumer_Task.Op2'access;
```

Finally, with the assignment above, we are able to impose that whenever *Op1* is called on *s1_Instance*, then the call is directed to the operation Op1 on the provided interface of the Consumer Sporadic Task, and from there it follows the correct delegation chain (redirection to the OBCS and reification of the request in the OBCS queue).

```
s0_Instance.Set_x(Consumer.Consumer_FC_Ref(s1_Instance'access));
```

Finally, we establish the binding between the Producer and the Consumer with the call above. In fact, it ensures that the call of *Op1* inside the OPCS of the Producer (*Producer_FC*) is a call to the provided interface of the Consumer sporadic task.

Conclusion

In this Ada Gems miniseries we described a set of Ravenscar-compliant code archetypes for the realization of recurrent patterns in real-time systems. We presented two basic patterns for the realization of cyclic and sporadic tasks, commented on their drawbacks, and showed how to improve them to realize sporadic operations with parameters and an example of complex queuing policy. Finally, we showed how to perform intertask communication.

The code archetypes we described were used for the code generation in the HRT-UML/RCM track of the EU-funded ASSERT project.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #104: Gprbuild and Configuration Files—Part 1

Author: Johannes Kanig, *AdaCore*

Let's get started…

Gem #65 introduced gprbuild, GNAT's program build tool that supports driving the build process of an Ada project with a project file. In particular, gprbuild is capable of building projects that use source files in programming languages other than Ada, whereas gnatmake only supports pure Ada projects.

Gprbuild needs two files: a project file (with the extension .gpr) defining characteristics of the project, such as the programming languages, source directories, compiler switches, main file(s), and so on, and a configuration file (with the extension .cgpr) describing the compilers to be used.

Project files are now widely used, most GNAT tools know how to deal with project files and how to take advantage of the information they contain, and many Ada projects are now described using project files. On the other hand, configuration files are not as easy to develop. Fortunately, the gprconfig tool, which is distributed with gprbuild, can autogenerate a configuration file which suits the most common situations. Even better, when one does not give a configuration file to gprbuild (using the --config" switch), gprbuild will automatically call gprconfig.

In this first Gem, we explain how to configure gprbuild to use a custom compiler.

A configuration file lists the compilers to be used for each language, as well as describing configuration options such as the needed compiler switches, the suffix for generated object files, the command-line switch for obtaining dependency information, and so on. Let's look at a simple example. While defining one's own compiler is useful mostly for nonstandard compilers, we will use GCC and the C programming language in our example, because most readers are familiar with this particular language and compiler. The configuration file gcc.cgpr looks like this:

```
configuration project GCC is
   package Compiler is
      for Driver ("C") use "/usr/gnat/bin/gcc";
      for Leading_Required_Switches ("C") use ("-c");
      for Object_File_Suffix ("C") use ".o";
      for Dependency_Switches ("C") use ("-MMD","-MF","");
      for Include_Switches ("C") use ("-I");
   end Compiler;

   package Naming is
      for Spec_Suffix ("C") use ".h";
      for Body_Suffix ("C") use ".c";
   end Naming;
end GCC;
```

This simple configuration file is sufficient to compile C files and is reasonably self-explanatory. The package "Compiler" defines the compiler executable, with required command-line switches, and the object file suffix. The package "Naming" introduces the usual naming scheme for C files.

Gprbuild has a built-in dependency mechanism that serves to avoid unnecessary recompilation when the relevant source files have not changed. It is of course desirable to use this mechanism. We achieve this here by setting the variable "Dependency_Switches", which gives the command-line options that trigger the generation of dependency files, in parallel to compilation. There is also a variable "Dependency_Driver" that can be set if one prefers to generate the dependency file independently from the invocation of the compiler. If your compiler supports the output of dependencies using Makefile syntax, then this mechanism is a simple way to obtain efficient incremental compilation for the given language.

The other parts of a configuration file, such as those that describe the linking process, will be described in a future Gem.

Now, given a C project, the following project file main.gpr is sufficient to describe it:

```
project Main is
   for Languages use ("C");
end Main;
```

As stated, gprbuild needs to be passed both the project file and the configuration file:

```
gprbuild --config=gcc.cgpr -P main.gpr
```

We have already described that if the "--config" switch is omitted, gprbuild generates a suitable configuration file automatically. It does so using the gprconfig tool. When executed, gprconfig reads a knowledge base describing a number of compilers with their options, their applicability to different platforms, and so on. Given this knowledge, gprconfig determines which compiler is available on the system. Gprconfig can then generate a configuration file containing only the descriptions of the available and relevant compilers.

The default knowledge base already contains complete descriptions of the GNAT and gcc compilers, and a few others. In the next Gem we will see the other parts of gprbuild that can be configured. In the third Gem of this series, we will see how to add a compiler to the knowledge base.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #105: Lady Ada Kisses Python—Part 1

Author: Emmanuel Briot, *AdaCore*

Let's get started…

The GNAT Components Collection (GNATCOLL) has included, since the beginning, a collection of packages to easily interface your Ada applications with scripting languages. This is the layer used in the GPS IDE to provide extensibility via the GPS shell or Python. The use with the GPS shell is just a toy we initially used to bootstrap the process, and was kept for backward compatibility only. On the other hand, Python is an extensively used object-oriented language that comes with its own run-time library, and can be easily extended in C or Ada.

This Gem will not go into the details of Python itself. There are excellent tutorials on the Internet. Instead, we will focus on the benefits that GNATCOLL provides over a direct interface via pragma Import, and show how to make your application scriptable in Python.

The package providing this support is called GNATCOLL.Scripts. As its name implies, it is meant to be a scripting-language-agnostic API. What this means is that your application will not know anything about Python or its API. Instead, you will export some classes and functions from Ada to GNATCOLL. The latter will then make sure that these functions are available for all the supported languages. Although currently we only support Python and the GPS shell, additional languages could conceivably be added in the future (such as JavaScript, Lisp, or Caml), and your application would automatically be scriptable through these.

GNATCOLL also does most of the memory management on your behalf. Python, for instance, uses reference counting to detect when an object can be freed. Lisp implementations, such as Scheme, generally use a form of garbage collection. But these are details you do not need to know about when you program via GNATCOLL.Scripts. In this package, the Ada types are controlled (and themselves use reference counting, as we detailed in an earlier Gem) and will automatically free memory when no longer used. This is of course less error prone, and will avoid a lot of memory leaks in your application.

In fact, GNATCOLL also provides a few minor extensions to the scripting package if you also program a GUI based on GtkAda. The latter (or rather the underlying gtk+ library) also uses its own reference counting. As a result, things can become really complex when you have an Ada object that's exported to Python, and this object is associated with one of the GUI elements in your application. Finding out when the memory is safe to deallocate requires a lot of care, but GNATCOLL takes care of it all on your behalf!

What are the benefits of interfacing with Python? Python, like a lot of scripting languages, has a rather high-level programming API. In particular, it makes the

construction of complex data structures relatively easy, both because it provides an important collection of such data structures that are fully integrated in the language, and also through its introspection and reflection capabilities. Our experience from GPS is that a lot of users (provided they are programmers, of course) will readily be able to understand Python script, and by using simple copy-paste will be able to quickly write their own scripts.

For GPS there are a number of user groups that have developed extensive Python modules to change the behavior of GPS and better integrate it in their own environment. Had we chosen to use Ada as the language for such extensions, it would have required users to have the full GPS development environment in order to build them, relink GPS (or create dynamically loaded libraries), and finally to test their changes. By comparison, the Python cycle is much faster: just edit and reload. Supportwise, it is also often convenient for us to provide a quick and short Python script to work around a GPS limitation and unblock customers, until we have time to do the proper work at the Ada level. Finally, there are features that are required by one customer, but which would make no sense for others; in such a case, the easiest thing to do is to implement it via a short Python script in coordination with the customer. If the Python API is kept stable, the script will continue to work from version to version of GPS.

In the second part of this Gem series, we'll show specific technical details of how to interface an application with Python using GNATCOLL.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

Gem #106: Lady Ada Kisses Python—Part 2

Author: Emmanuel Briot, *AdaCore*

Let's get started…

The first part of this Gem described why having a Python interface might provide an effective way to customize and extend your application. It also highlighted how the GNAT Components Collection takes care of a good part of the work. This Gem will now go into the technical details of how you can use GNATCOLL.Scripts in practice. The GNATCOLL documentation provides additional details, so please refer to it if you need more information (see GNATColl: GNAT Reusable Components).

Your application needs to indicate which scripting languages it wants to support. Handles to those languages are stored in a global variable called a Scripts_Repository (which your application provides, so it could be stored in your own record type, or wrapped in a protected object, etc.).

```
with GNATCOLL.Scripts.Python;
use GNATCOLL.Scripts, GNATCOLL.Scripts.Python;

declare
   Repo : Scripts_Repository := new Scripts_Repository_Record;
begin
   Register_Python_Scripting (Repo, "GPS");
end;
```

The second parameter on the last line is the name of the Python module that your application exports. This example is extracted from GPS, where all functions and classes are available as GPS.Console, GPS.Logger, etc. This parameter is the namespace in which your exports will go.

The next step is to register some of the standard classes that are needed by GNATCOLL itself. The most important of these is the Console class, which provides input/output between your application and Python. In many cases you do not want to allow Python to output to stdout; for instance, if you are writing a GUI application you would have an interactive Python console for this. As a result, the standard Python input/output will be redirected to instances of the Console class.

```
Register_Standard_Classes (Repo, "Console");
```

We now need to define how a console behaves at the Ada level (either to the usual stdin and stdout, or to a GUI window for instance). We will not go into the details of creating custom consoles, but GNATCOLL comes with examples for the two usual cases in its examples/textconsole.ads and examples/gtkconsole.ads files.

```
Console := GtkConsole.Create (...);  --  See the examples directory
```

```
Set_Default_Console
  (Lookup_Scripting_Language (Repo, "python"), Virtual_Console
(Console));
```

At this point, your application can show a GUI window to interact with Python, in which users can write usual Python commands. But your application still hasn't exported anything useful.

Let's take a simple example. We want to export from Ada a function that performs the addition of two integers (of course, this is already provided by Python, but this is just an example). We first need to declare what we are exporting:

```
Register_Command
   (Repo,
    Command => "add",
    Params  => (1 => Param("p1"), 2 => Param("p2"),
                3 => Param("p3", Optional => True)),
    Handler => Handler'Access);
```

This indicates that from Python we will be able to make calls such as:

```
  n = add(p1=23, p2=45)
  n = add(p2=45, p1=23)   # order of parameters is irrelevant
  n = add(23, 45, 67)     # three parameters
```

The first version shows the use of named parameters in Python (one of the nice features it inherited from Ada). These parameters can be specified in any order in Python, and GNATCOLL will automatically reorder them so that your application always accesses "p1" as the first parameter and "p2" as the second parameter.

As you can see in the Ada code, there is no reference to Python. In fact, "add" will be available in all registered languages (Python here, but also potentially the GPS shell, and in the future other languages). If new languages are added in GNATCOLL, your code will not need any change to benefit from these.

We now need to provide the implementation:

```
procedure Handler (Data : in out Callback_Data'Class; Command : String)
is
   P1, P2 : Integer;
begin
   if Command = "add" then
      P1 := Nth_Arg (Data, 1);
      P2 := Nth_Arg (Data, 2);
      P3 := Nth_Arg (Data, 3, 0);  --  Default value is 0

      Set_Return_Value (Data, P1 + P2 + P3);
   end if;
end Handler;
```

Again, this code is independent of Python. We always access "P1" as the first parameter (through the call to Nth_Arg), and GNATCOLL will automatically take care of reordering the parameters if the user has specified them in a different order.

GNATCOLL will automatically raise a Python exception if the user calls "add" with an incorrect number of parameters. Note that it's also possible to export functions with optional parameters (P3 in our example). Nth_Arg can then be used to specify the default value for parameter.

Also worth noting is that Nth_Arg will raise an exception if the corresponding parameter has an incorrect type. There exist several variants of Nth_Arg (for strings, integers, booleans, and a few other types). In our case, if the user calls "add" with a string, Nth_Arg will raise an Ada exception that your application can handle. If your application does not catch the exception, it will be propagated to Python.

The example in this Gem has a limited scope, but highlights some of the fundamental features and services that GNATCOLL offers on top of Python. Once GNATCOLL has been properly initialized, exporting new commands requires little additional glue code. Exporting classes and functions is very similar to what the example described. For more information, see the online GNATCOLL documentation at GNATCOLL: Embedding Script Languages.

At this stage, the most difficult task is to define a clear Python API to your application, one that users can understand relatively easily, and yet that can be extended in the future by exporting even more capabilities from Ada, without breaking the whole design of the API.

Related Source Code

Ada Gems example files are distributed by AdaCore and may be used or modified for any purpose without restrictions.

SIGAda
The ACM Special Interest Group on Ada

Ricky E. Sward
Chair, ACM SIGAda
1155 Academy Park Loop
Colorado Springs, CO

August 30, 2012

Dear SIGAda Members,

It's hard to believe that almost four years have passed since the current officers were elected and took office. Over the next few months, SIGAda will hold officer elections and the new Executive Committee will take office in the summer of 2013. If you are interested in participating in SIGAda as one of the elected officers, please contact John McCormick, our past Chair, about how to volunteer. John will be selecting two candidates for each position for the upcoming elections.

In our last viability review, the ACM SIG Governing Board commented on how many elected officer positions SIGAda currently has. There are other SIGs that have about the same number of members as SIGAda, and they have only three or four elected officers. SIGAda currently has six: Chair, Vice Chair for Meetings and Conferences, Vice Chair for Liaison, Secretary, Treasurer and International Representative. At the last Extended Executive Committee (EEC) meeting held in Denver during SIGAda 2011, the EEC discussed a possible reorganization of the Executive Committee to include fewer elected officer positions.

We are proposing to combine the current Vice Chair positions into one position and to also combine the Secretary and Treasurer positions into one position. This reorganization would reduce the number of elected SIGAda officers from six to four: Chair, Vice Chair, Secretary-Treasurer and International Representative. The justification for this change is that it will reduce the number of volunteers that SIGAda needs to find to fill the officer positions. It also enables other volunteers to serve in Conference Chair and Local Arrangements chair positions to support the SIGAda annual conferences.

This proposed reorganization requires a change to the SIGAda Bylaws. The Bylaws were originally approved by ACM in 1984 and revised under Currie Colket in 2005. The proposed revised bylaws are published in this copy of Ada Letters for your review. Here is a detailed summary of the proposed changes to the SIGAda Bylaws.

1) Article 4 – Officers: Combine the current Vice Chair for Meetings and Conferences position with the Vice Chair for Liaison. The new name of this position is Vice Chair. The duties have been combined and reworded appropriately.
2) Article 4 – Officers: Combine the current Secretary position with the Treasurer position. The new name of this position is Secretary-Treasurer. The duties have been combined and reworded appropriately.
3) Article 6 – Vacancies and Appointments: Change the reference to the Vice Chair for Meetings and Conferences to reflect the new name, Vice Chair.

Over the next few months, ACM will inform the SIGAda membership about these changes to the SIGAda Bylaws. We anticipate that these changes will be considered minor changes by ACM and that we can approve the new reorganization before the upcoming elections.

If you have any questions or comments about the Bylaws changes, please feel free to contact me at ricky.sward@acm.org. Thank you for your consideration.

Cheers,

Ricky E. Sward
SIGAda Chair

SIGAda Annual Report
July 1, 2011 - June 30, 2012

June 30, 2012

SIGAda Awards

Started in 1994, the ACM SIGAda Awards recognize individuals and organizations that have made outstanding contributions to the Ada community and to SIGAda. The Outstanding Ada Community Contribution Award is given for broad, lasting contributions to Ada technology and usage. The Distinguished Service Award is given for exceptional contributions to SIGAda activities and products.

This year the Outstanding Ada Community Contribution Awards were awarded to Stephen Michell and Tullio Vardanega.

Stephen Michell - Member of the ISO/IEC JTC 1/SC 22/WG 9 and Ada Rapporteur Group (ARG). Frequent participant in the International Real-Time Ada Workshop (IRTAW). Involved with the Ada High Integrity Rapporteur Group (HRG). 18 citations for papers and works involving Ada. Currently the Canadian Head Of Delegation for WG23, Software Vulnerabilities.

Tullio Vardanega - Member of the ISO/IEC JTC 1/SC 22/WG 9 and Ada Rapporteur Group (ARG). Frequent participant in the International Real-Time Ada Workshop (IRTAW). Served as Editor-in-Chief of Ada-Europe's Ada User Journal (AUJ). Served on the Ada-Europe Board. Served as Program Co-Chair for four Ada-Europe conferences. Served as Conference Chair for a memorable conference. Served as president of Ada-Europe.

This year the Distinguished Service Award was awarded to Alok Srivastava.

Alok Srivastava - Editor, Ada Letters. Alok has done an outstanding job publishing three issues of Ada Letters each year, which is one of the key member benefits advertised by the SIGAda organization. Two time Conference Chair (2007 and 2010). The 2007 conference was the most financially successful conference in recent history. Vice Chair for Meetings and Conferences since 2009. He has done an outstanding job overseeing the organization of the SIGAda conferences for the past three years. He has also stepped in several times for the SIGAda Chair to run bi-weekly meetings and to attend ACM SIG Governing Board meetings.

Significant Papers published in proceedings

This year's conference included three outstanding keynote speeches. The keynote speakers presented on the following topics:

Grady Booch. IBM Fellow. Chief Scientist for Software Engineering, IBM Research.
Everything I Know I Learned from Ada

Martin Carlisle, Ph.D. US Air Force Academy. *Why I Came Back To Ada*

Jim Rogers. MEI Technologies, Inc. *Software Safety, and Related Language Considerations*

This year's conference included an outstanding panel session that was very well received by the attendees:

Panel: How to Make Ada Go "Viral". Chair: JP Rosen (AdaLog), with Brad Moore (General Dynamics Canada), David Sauvage (AdaLabs, Mauritius), Tucker Taft (Sofcheck)

There were several outstanding papers in the conference this year with equally outstanding presentations. For example:

Stack Safe Parallel Recursion with Paraffin by Brad Moore (General Dynamics, Canada)

Software Vulnerabilities Precluded by SPARK by Paul E. Black (National Institute of Standards and Technology), Chris E. Dupilka (US DoD), F. David Jones and Joyce Tokar (Pyrrhus Software)

Enhancing SPARK's Contract Checking Facilities Using Symbolic Execution by John Hatcliff, Jason Belt (Kansas State University), Patrice Chalin (Concordia University), David Hardin (Rockwell Collins Advanced Technology Center), and Xianghua Deng (Google, Inc.)

An Ada Design Pattern Recognition Tool for AADL Performance Analysis by V. Gaudel, F. Singhoff, A. Plantec, S. Rubini (University of Brest, France), P. Dissaux (Ellidiss Software), and J. Legrand (Ellidiss Software)

Overall, the papers being submitted to the SIGAda conference continue to be of high quality.

Significant Programs that provided a springboard for further technical efforts

A formal liaison exists between SIGAda and WG9. ISO/IEC JTC1/SC22 WG9 is that body of international representatives responsible for the maintenance and evolution of the Ada International Standard. The National Bodies represented on WG9 are Belgium, Canada, France, Germany, Italy, Japan, Switzerland, the United Kingdom, and the United States.

In March 2007 the ISO (the International Organization for Standardization) in Geneva, Switzerland announced the formal completion of the process to revise the Ada 95 language, with the publication of the Ada 2005 standard — officially named ISO/IEC 8652:1995/Amd 1:2007. This announcement culminates a collaborative international effort under ISO's Ada Working Group (WG9) to enhance the 1995 version of the Ada language.

At least one SIGAda Officer participates and represents the membership at the WG9 meetings held twice each year.

Innovative Programs which provide service to some part of our technical community

Since 1994 SIGAda has conducted an "Ada Awareness Initiative". Its centerpiece has been our SIGAda professional booth display unit in exhibition halls at important software engineering conferences. This lets folks know that Ada is very much alive and a sound part of any software engineering effort having real-time, high integrity, high-assurance, and highly distributed requirements. We brought the booth to the SIGCSE conference this year providing good visibility for SIGAda to the Computer Science educational community. We decided not to take the booth to the Software and Systems Technology Conference (SSTC) due to declining attendance at the conference.

Via this exhibiting, SIGAda sustains Ada visibility ("name recognition"), provides various Ada-advocacy materials and makes available Ada experts (our booth staff volunteers) who can intelligently answer questions, provide pointers and help, and debunk the misinformation about Ada that many attendees at these shows have. This program continues to be extremely successful and viewed as a highly important thrust by the SIGAda membership.

Summary of key issues to deal with in the next 2-3 years

One of the key issues for SIGAda is a proposed revision of the bylaws that includes a reorganization of the Executive Committee. There are currently six elected officers including the International Representative, who is elected by the Ada Europe members. Due to the size of SIGAda, we propose a smaller Executive Committee that consists of just four elected officers: Chair, Vice Chair, Secretary/Treasurer, and International Representative. Since the International Representative will still be elected by Ada Europe members, this new organization includes only three officers that will be elected by the SIGAda membership. This simplifies the organization and eases the burden of finding willing candidates to fill the officer positions.

Another key issue is continuing to host a financially successful conference. Last year the conference in Denver had a surplus in revenues of $9180.00. This reversed the trend of holding conferences that had a loss in revenues, and was we will strive to continue this trend. We will continue to encourage our SIGAda members to participate in and to attend the conference.

We decided to rename the SIGAda annual conference in order to focus on a niche in the safety critical, high integrity area of Computer Science. The SIGAda 2012 annual conference is called the High Integrity Language and Technology (HILT) conference. We already have confirmed that several well-known researchers in this area will attend and give keynote speeches, including Barbara Liskov, Nancy Leveson, and Guy Steele.

We will continue to publish three issues of the Ada Letters journal and seek participation in the form of contributing articles and papers.

Ricky E. Sward
Chair ACM SIGAda

REUSABLE SOFTWARE COMPONENTS

Trudy Levine
Fairleigh Dickinson University
Teaneck, NJ 07666
levine@fdu.edu
http://alpha.fdu.edu/~levine/reuse_course/columns

This column consists of our August 2012 listing of sources for reusable software components. All information is obtained directly from web sites or from parties affiliated with the sites. As always, no recommendations or guarantees are implied.

For anyone interested, I have hard copies of Ada Letters going back to 1988, available for reuse.

We appreciate comments, corrections, and suggestions from our readers.

~~~~~~

The Ada-Belgium Archive

One of the aims of the Ada-Belgium organization is to disseminate Ada-related information. So, in addition to the organization of seminars, workshops, etc., and the management of two mailing lists, it also has set up an Ada archive which enables everyone interested to consult and download a large variety of Ada software and documents using a server in Belgium.
Key items include:
* A disk copy of the last version of the Ada and Software Engineering Library (ASE2, a 2 disk CD-ROM set).
<ftp://ftp.cs.kuleuven.be/pub/Ada-Belgium/cdrom/index.html>
* A complete archive of the last public GNAT distribution that uses the GNAT Modified General Public License (3.15p).
<ftp://ftp.cs.kuleuven.be/pub/Ada-Belgium/mirrors/gnu-ada/>
* A directory with Free Ada Software provided by Belgian Ada users.
<http://www.cs.kuleuven.be/~dirk/ada-belgium/software/>
New - a link to Rob Veenker's description of how to use an Ada application on an Android device:
http://rveenker.home.xs4all.nl/Ada%20on%20Android.html
The Ada-Belgium archive is primarily intended for the Belgian Ada community, but anyone interested is welcome to use it.
http://people.cs.kuleuven.be/~dirk.craeynest/ada-belgium/ (last update 2012)

~~~~~~

Ada Class Library

ACL is an object oriented library for Ada.Text search and replace. Scripting (small tool programs). CGI scripts. Execution of external programs (incl. I/O redirection). Garbage Collection. Extended Booch Components. CD-Recorder.
An AdaCL release for Ada 2005 is included.

http://sourceforge.net/projects/adacl/ or **http://adacl.sourceforge.net (last update 2011)**

~~~~~~
~~~~~~

Ada Core

AdaCore provides open source tools and expertise for the development of mission-critical, safety-critical, and security-critical software. AdaCore's flagship products are the GNAT Pro and SPARK Pro development environments and the CodePeer automatic code reviewer and validator. The GNAT technology is the first to support all three ISO standards of the Ada programming language - Ada 83, Ada 95, Ada 2005, as well as supporting the key features of Ada 2012. GNAT Pro also comes with Frontline Support (provided by the developers of the toolset) and expert Ada consulting.

The GNAT technology includes:

- GNAT Programming Studio IDE
- Full Ada Compiler (Ada 83/Ada 95/Ada 2005/Ada 2012) Utilities for Analysis, Testing and Code Navigation Visual Debugger Libraries and Bindings Runtime Profiles
- GNAT Pro High-Integrity Family of products supporting safety and security standards such as DO-178B and MILS Support for over 70 native and cross platforms including Unix, Linux, Windows, .NET, the JVM, bare boards, VxWorks 5/6/653/MILS, LynxOS, and PikeOS

Add-on technologies:

- GNATbench - Plug-In for Eclipse (GNAT Pro) GNATstack - Stack Analysis Tool (GNAT Pro) Ada Web Services - Web-Based Technologies GtkAda - Intuitive GUI Builder and Extensive Widget Set XML/Ada - XML library GLADE Ada 95 Distributed Systems Annex Implementation PolyORB - Middleware ASIS-for-GNAT - Ada Semantic Analysis. CodePeer - automatic code review and robustness validation. SPARK Pro - code verification, based on information-flow analysis and theorem-proving.
- GNAT AJIS - allows Ada projects to integrate Java code and allows Ada projects to develop code for teams developing in Java, C and C++ binding generators - generation of bindings for C and C++ headers.
- GNATcheck - qualified coding standard checker

The GNAT Academic Program (GAP) was created to help bring Ada to the forefront of university study. It includes a comprehensive toolset and support package designed to give educators the tools they need to teach Ada.
Free Software developers and students can download GNAT GPL from
http://libre.adacore.com/libre
http://www.adacore.com/home/academia
or contact: gap-contact@adacore.com (Site updated 2012)
~~~~~~

Ada-Europe

Ada-Europe is an international organization, set up to promote the use of Ada. Ada-Europe represents European interests in Ada and Ada-related matters. Member organizations include:
Ada-Belgium, Ada-Denmark, Ada-Deutschland, Ada-France, Ada-Spain, Ada in Sweden, and Ada in Switzerland. All of these organizations maintain web sites with available resources.
See: **http://www.ada-europe.org**
http://www.ada-europe.org/resources/online/ for Annotated Ada 2012 Language Reference Manual

~~~~~~

Ada IC

The Ada Information Clearinghouse has been providing free information about Ada and software engineering for over fifteen years. Sponsored by the Ada Resource Assoc. (http://www.adaresource.com) a consortium of Ada tool vendors and developers, the AdaIC maintains close contact with the Ada community in order to obtain the latest information on a variety of topics.

Visit their website, **http://www.adaic.org**, to see the latest in news, implementation guidelines, compilers and tools, reusable Ada code, education and training, Ada successes, and lessons learned by software developers. The site remains current with many resources targeted for Ada 2012. Several blogs are maintained to continue conversion on listed topics.

The Ada-wide search engine indexes all known Ada content (more than 76,000 pages according to Randy's last count). General search engines, such as Google, have too many references to the term "Ada" to make them practical for the purposes of the Ada community.

Please send any news you have on Ada to **<news@adaic.org>.** The Ada News of the AdaIC sometimes transmits press releases about the programming language to about 500 technical journalists and editors, as well as citing it on the AdaIC Website, as a free service to its users.

A comprehensive collection of Ada articles, reports, textbooks, videos, and CD-ROMS is available for browsing on-line through the AdaIC website. Users may access older components at the Virtual Library: **http://archive.adaic.com** (updated 7/12)
Reusable software components are available at
http://www.adaic.org/ada-resources/tools-libraries/
~~~~~~

AJPO

The Ada Joint Project Office was closed on October 1998. For information on the AJPO see
http://sw-eng.falls-church.va.us/ajpofaq.html
http://sw-eng.falls-church.va.us/ajpo_databases/products_tools1.html
~~~~~~

Adalog

Adalog offers Ada utilities, Ada components, and Adapplets. These can be freely used and modified for any purpose, under the GMGPL license. Functions include Protection, Debugging, and OS_Services, among others.

The site also contains Adasubst/Adadep programs which are useful utilities for rearranging Ada programs, and AdaControl, a powerful utility for checking and enforcing style and coding rules. AdaControl is a free (GMGPL) tool that detects the use of various kinds of constructs in Ada programs. Its first goal is to control proper usage of style or programming rules, but it can also be used as a powerful tool to search for use (or non-use) of various forms of programming styles or design patterns. Searched elements range from very simple, like the occurrence of certain entities, declarations, or statements, to very sophisticated, like verifying that certain programming patterns are being obeyed.
Since it is GMGPL, all of its parts can be reused for any purpose.

These programs are built on top of ASIS and include valuable packages providing higher level queries for ASIS (package Thick_Queries). For example, look for the function called "Full_Name_Image," which returns the unique name of any Identifier.

In addition, there is sc_timer, the Session Chair universal clock, which is very useful to those who have to chair a session, and a demo of GTK-Ada.

SEE: **http://www.adalog.fr/** (site updated 2012)
Ada components available at **http://www.adalog.fr/compo1.htm**
~~~~~~
~~~~~~

AdaPower

AdaPower's website has been entirely redone in Ada, using GRAW, a rapid agile web development framework, to maintain their repository of Ada information, links to resources, source code examples, and packages for reuse.

AdaPower contains:

- Articles and Links to Ada Related Topics, Ada learning materials, and people in the Ada on-line community
- The Ada Source Code Treasury
  Source code examples of using Ada and Ada related bindings and tools for both beginner and advanced students of Ada
- Packages for Reuse
  An extensive repository of categorically arranged packages for download and links to packages available for reuse on the internet

http://www.adapower.com/ (Site last updated 2012)
http://www.adapower.com/index.php?Command=Packages&Title=Packages+for+Reuse
http://www.adapower.com/index.php?Command=Class&ClassID=AdaLibs&Title=Ada+Libraries

~~~~~~

Ada Structured Library Version 1.4

Ada structured Library is a set of general containers and utilities. The library is licensed under the same license as GNAT (see GNU, below), which is GPL but is modified to allow inclusion into a program without bringing the whole program under the GPL.

The utilities include some things lacking in Ada95, including:
* Abstract I/O - allows the I/O user and the I/O to be decoupled, so you can do file I/O, socket I/O, serial I/O telnet, etc. by changing the I/O object the user references. Includes many functions of Ada.Text_IO.
* Calendar - Full-featured time and calendar manipulation.
* Telnet - A general telnet library implemented over sockets.
* Command processor - Does string tokenizing and command processing over Abstract I/O.
* A set of general-purpose containers, including Lists, Vectors, Trees, Graphs, and a Btree, with lots of options.

See: http://adasl.sourceforge.net/
http://sourceforge.net/projects/adasl (Site updated 2009)

~~~~~~

Booch Components

The Ada 95 Booch Components began in late 1994 when David Weller began a port of Grady Booch's C++ components to Ada95. They have since been taken over by Simon Wright and at this time, include implementations of bags, collections, dequeues, graphs, lists, maps, queues, rings, sets, stacks, and trees. These include definite and indefinite types, bounded and unbounded implementations, dynamic and static storage allocations. Filtering and sorting operations are supported. The Containers are compatible with both Ada 95 and Ada 2005. Backward compatibility with Ada 95 is retained.
http://sourceforge.net/projects/booch95/
http://sourceforge.net/projects/booch95/develop (Site updated 2012)

CONTACT: Simon Wright (simon@pushface.org) or Martin Krischik (martin@krischik.com)

DMOZ

DMOZ is a free, open directory project, with Ada components submitted and maintained by volunteers. The site links to several of the items that are listed elsewhere in this column, as well as many more, including Ada Lexer, a lexical analyzer written in Ada that recognizes Ada 2012 reserved words.

http://dmoz.org/Computers/Programming/Languages/Ada/Bindings_and_Libraries/ (updated 2011)
~~~~~~

Free Software Foundation

The Free Software Foundation is dedicated to eliminating restrictions on people's right to use, copy, modify, and redistribute computer programs. It promotes the development and use of free software and its documentation in all areas using computers. Specifically, it is maintaining a complete, integrated software system named "GNU". ("GNU" is pronounced "guh-new" and stands for "GNU's Not Unix".)

The word "free" in "Free Software Foundation" refers to freedom, not price. You may or may not pay money to get GNU software, but regardless you have specific freedoms once you get it: the freedom to copy a program and give it away to your friends and co-workers; and the freedom to change a program as you wish, by having full access to source code. You can study the source and learn how such programs are written. You may then be able to port it, improve it, and share your changes with others. If you redistribute GNU software you may charge a distribution fee or give it away.

For the Free Software Definition, see: www.gnu.org/philosophy/free-sw.html

What is Copyleft?

The simplest way to make a program free is to put it in the public domain, uncopyrighted. But this permits proprietary modifications, denying others the freedom to use and freely redistribute improvements; it is contrary to the intent of increasing the total amount of free software. To prevent this, copyleft uses copyrights in a novel manner. Typically copyrights take away freedoms; copyleft preserves them. It is a legal instrument that requires those who pass on programs to include the rights to use, modify, and redistribute the code; the code and rights become legally inseparable.

The copyleft used by the GNU Project is made from the combination of a regular copyright notice and the "GNU General Public License." (www.gnu.org/copyleft/gpl.html) GPL is a copying license which basically says that you have the aforementioned freedoms. An alternate form, the "GNU Lesser General Public License" applies particularly to certain GNU libraries. This license permits linking the libraries into proprietary executables under certain conditions.

See www.gnu.org/copyleft/copyleft.html
www.gnu.org/licenses/licenses.html

GNAT is listed in the Free Software Directory, which catalogs useful free software that runs under free operating systems, particularly the GNU operating system and its GNU/Linux variants. The GNAT Technology includes the implementation of the ASIS standard (Ada Semantic Interface Specification), GtkAda to build portable and efficient GUIs in Ada, AWS (Ada Web Server) the framework to develop Web-based applications in Ada, the XML/Ada library to process XML streams in Ada, GLADE to develop distributed applications following the Ada Distributed Systems Annex standards, and PolyORB to develop distributed applications following the CORBA standard.

The GNAT GPL 2012 Edition, which is available free of charge from libre.adacore.com/, is licensed for Free Software development under the terms and conditions of the GNU General Public License..

For more information visit the following links:

GNAT Pro: www.adacore.com/gnatpro/
http://directory.fsf.org/wiki/GNAT (site updated 2010)

Free Software Foundation, Inc.	+1 617 542 5942 x 23
51 Franklin Street, Fifth Floor	+1 617 542 2652 (fax)
Boston, MA 02110-1301	email: info@fsf.org
See: **http://www.fsf.org**	http://www.gnu.org
~~~~~~

~~~~~~

Kazakov Objects

Dmitry Kazakov maintains a web site of free Ada components. The license is GM GPL, where appropriate. The library conforms to both Ada 95 and Ada 2005 language standards and includes:

1. Objects and handles (smart pointers)
2. Persistency
3. Sets and maps
4. Unbounded arrays
5. Unbounded arrays of pointers
6. Stacks
7. Pools
8. Doubly-linked networks
9. Graphs
10. Lock-free structures
11. Locking synchronization primitives
12. Parsers
13. Cryptography
14. Numerics
15. Miscellany
16. Packages
17. Installation
18. Changes log

See: www.dmitry-kazakov.de/ada/components.htm
www.download25.com/simple-components-for-ada-download.html (site updated 2012)
~~~~~~

Leake Components

Stephen Leake maintains the following Ada components:
com ports: An Ada binding, based on Win32Ada, to the Win32 com port facilities.
Auto_Text_IO: automatically generates Text_IO packages for Ada packages
Stephe's Ada Library: another entry in the Standard Ada Library sweepstakes

A large part of SAL provides math operations for kinematics and dynamics of masses in 3 dimensional space. Cartesian vectors, quaternions, orthonormal rotation matrices, moments of inertia, forces, acceleration, velocity are supported, in 3 and 6 degrees of freedom (translation and rotation). This library has been used for both robotics and satellite simulation.

http://stephe-leake.org/
http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html
http://stephe-leake.org/ada/arm.html contains, in tar gzip format, an info version of the Ada 2005 and 2012 Reference Manuals

Matreshka

Matreshka is an Ada framework to help develop information systems. It includes:

- League --- provides support for localization, internationalization and globalization; including:
    - unbounded form of string of Unicode characters; cursors to iterate other characters and grapheme clusters; advanced locale tailored operations such as case conversion, case folding, collation, normalization;
    - calendars and calendrical calculations;
    - regular expression engine with Perl-style syntax and Unicode extensions;
    - text codec to convert data streams into/from internal representation;
    - message translator to translate messages into natural language which is selected by user;
    - access to command line arguments and environment variables as Unicode encoded strings.
    - persistent application settings to manage application settings, supports INI files and Windows Registry.
- XML processor --- provides capability to manipulate with XML streams and documents; including:
    - SAX reader to read XML streams and documents; it supports both XML1.0/XML1.1 specifications as well as corresponding Namespaces in XML specifications;
    - SAX writer to generate XML streams and documents from application.
- Web framework
    - FastCGI module assists with developing server side applications completely in Ada and using them with standard HTTP servers.
- SQL database access provides simple generic API to access to SQL databases. Supported databases:
    - _Oracle, _PostgreSQL , and_SQLite3

Two new features were added to Matreshka last year:

- Support for Firebird/Interbase database;
- The Ada Modeling Framework now provides implementation of OMG's Meta Object Facility (MOF) written completely in Ada. Extension modules are provided to analyze/modify:
- UML models and their extensions:
- MOF Extensions to support meta-modeling
- OCL models
- UML Testing Profile to support Model-Driven Testing

http://forge.ada-ru.org/matreshka (site updated 2012)

~~~~~~

QtAda

QtAda is an Ada2005 language binding to the Qt libraries and a set of useful tools. Qt is a cross-platform C++ development framework developed and supported by Qt Software. QtAda supports Qt version 4.4 and later. QtAda supports the development of a cross-platform powerful graphical user interface completely in Ada 2005. QtAda applications will work on most popular platforms — Microsoft Windows, Mac OS X, Linux/Unix — without any changes and platform specific code. QtAda applications use native look and feel (and even utilize user Control Panel settings) on every supported platform.

QtAda is not just a binding to the existent Qt widgets. It also allows the development of your own widgets and integrates them into the Qt Designer for high speed visual GUI development. QtAda uses native thread safe signal/slot mechanism and provides full transparent integration with Ada tasks. QtAda provides support for application localization/internationalization, including message translations, local specific character and string processing, date/time and numeric formatting.

QtAda Academic Edition is suitable for educational purposes and includes free support for professors and teachers.

See**: http://www.qtada.com/en/index.html**

Email: sales@qtada.com (Site updated 2010)
~~~~~~

~~~~~~

USAFA

Professor Martin Carlisle at the US Air Force Academy continues to develop free software for use by the computer science community. Although previously known for tools specifically for Ada programmers (in particular A#, AdaGIDE, and RAPID), his more recent development has targeted the computer science education and computer security audiences. The newest tools, RAPTOR and IRONSIDES, have Ada inside and are developed using AdaGIDE, GNAT, SPARK Ada, and A#. RAPTOR is a flowchart-based programming environment useful for teaching introductory computer science and is taught in at least 22 countries.

IRONSIDES is an authoritative DNS server implemented in SPARK Ada using formal methods to prove the absence of many major categories of security vulnerabilities.

More information on these projects can be found at:

http://ironsides.martincarlisle.com
http://raptor.martincarlisle.com
http://adagide.martincarlisle.com
http://www.martincarlisle.com/ada_stuff.html
http://asharp.martincarlisle.com
http://rapid.martincarlisle.com

CONTACT: Martin C. Carlisle, Professor of Computer Science, US Air Force Academy
carlislem@acm.org
~~~~~~

ACM SIGAda Annual International Conference

High Integrity Language Technology
HILT 2013
Call for Technical Contributions

Developing and Certifying Critical Software

Pittsburgh, Pennsylvania, USA
Fall of 2013 [Mid-October to Mid-November]

Sponsored by ACM SIGAda
SIGAda.HILT2013@acm.org
http://www.sigada.org/conf/hilt2013

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. To this end we are soliciting technical papers, experience reports (including experience in teaching), and tutorial proposals on a broad range of relevant topics.

POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

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

KINDS OF TECHNICAL CONTRIBUTIONS

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM *Ada Letters*. The Proceedings will be entered into the widely consulted ACM Digital Library accessible online to university campuses, ACM's 100,000 members, and the software community.

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

EXPERIENCE REPORTS present timely results and "lessons learned". Submit a 1-2 page description of the project and the key points of interest. Descriptions will be published in the final program or proceedings, but a paper will not be required.

PANEL SESSIONS gather groups of experts on particular topics. Panelists present their views and then exchange views with each other and the audience. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation and, if selected, a subsequent abstract for a 30-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for ACM *Ada Letters*.

WORKSHOPS are focused sessions that allow knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. Workshop proposals, up to 5 pages in length, will be selected based on their applicability to the conference and potential for attracting participants.

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

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

Submission	***Deadline***	***Send to***
Technical articles, extended abstracts, experience reports, panel session proposals, or workshop proposals	**June 29, 2013**	**Tucker Taft,** Program Chair `taft@adacore.com`
Industrial presentation proposals	**August 1, 2013** (overview) **September 30, 2013** (abstract)	
Tutorial proposals	**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** (`gicca@adacore.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 contact the Conference Chair (**Jeff Boleng**, `jlboleng@SEI.CMU.EDU`), SIGAda's Vice-Chair for Meetings and Conferences (**Alok Srivastava**, `alok.srivastava@tasc.com`), or SIGAda's Chair (**Ricky E. Sward**, `rsward@mitre.org`).

IRTAW 2013

The 16th International Real-Time Ada Workshop - IRTAW 2013

http://www.cs.york.ac.uk/~andy/IRTAW2013/

17-19 April 2013

Kings Manor, York, England

Call for Papers

Since the late Eighties the International Real-Time Ada Workshop series has provided a forum for identifying issues with real-time system support in Ada and for exploring possible approaches and solutions, and has attracted participation from key members of the research, user, and implementer communities worldwide. Recent IRTAW meetings have significantly contributed to the Ada 2005 and Ada 2012 standards, especially with respect to the tasking features, the real-time and high-integrity systems annexes, and the standardization of the Ravenscar profile.

In keeping with this tradition, the goals of IRTAW-16 will be to:

- review the current status of the Ada 2012 Issues that are related with the support of real-time systems;
- examine experiences in using Ada for the development of real-time systems and applications, especially – but not exclusively – those using concrete implementation of the new Ada 2012 real-time features;
- report on or illustrate implementation approaches for the real-time features of Ada 2012;
- consider the added value of developing other real-time Ada profiles in addition to the Ravenscar profile;
- examine the implications to Ada of the growing use of multiprocessors in the development of real-time systems, particularly with regard to predictability, robustness, and other extra-functional concerns;
- examine and develop paradigms for using Ada for real-time distributed systems, with special emphasis on robustness as well as hard, flexible and application-defined scheduling;
- consider the definition of specific patterns and libraries for real-time systems development in Ada;
- identify how Ada relates to the certification of safety-critical and/or security-critical real-time systems;
- examine the status of the Real-Time Specification for Java and other languages for real-time systems development, and consider user experience with current implementations and with issues of interoperability with Ada in embedded real-time systems;
- consider the lessons learned from industrial experience with Ada and the Ravenscar Profile in actual real-time projects;
- consider the language vulnerabilities of the Ravenscar and full language definitions.

Participation at IRTAW-16 is by invitation following the submission of a position paper addressing one or more of the above topics or related real-time Ada issues. Alternatively, anyone wishing to receive an invitation,

but for one reason or another is unable to produce a position paper, may send in a one-page position statement indicating their interests. Priority will, however, be given to those submitting papers.

Submission Requirements

Position papers should not exceed ten pages in typical IEEE conference layout, excluding code inserts. All accepted papers will appear, in their final form, in the Workshop Proceedings, which will be published as a special issue of Ada Letters (ACM Press). Selected papers will also appear in the Ada User Journal.

Please submit position papers, in PDF format, to the Program Chair by e-mail: alan.burns@york.ac.uk

Important Dates

- Paper Submission: **1 February, 2013**
- Notification of Acceptance: 1 March, 2013
- Confirmation of Attendance: 14 March, 2013
- Final Paper Due: 1 April, 2013
- Workshop: April 17-19, 2013

Program Chair: Alan Burns, University of York

Workshop Chair: Andy Wellings, University of York

Program Committee Members: Mario Aldea Rivas, John Barnes, Ben Brosgol, Alan Burns, Michael Gonzàlez Harbour, 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.

Sponsors

Call for Papers
18th International Conference on Reliable Software Technologies
Ada-Europe 2013
10-14 June 2013, Berlin, Germany

http://www.ada-europe.org/conference2013

Special Interest Group Ada of the German Informatics Society

Conference and Program Co-Chairs

Hubert B. Keller
Karlsruhe Institute of Technology
hubert.keller@kit.edu

Erhard Plödereder
University of Stuttgart
ploedere@iste.uni-stuttgart.de

Tutorial Chair

Jürgen Mottok
Regensburg University of Applied Sciences
Juergen.Mottok@hs-regensburg.de

Industrial Chair

Jørgen Bundgaard
Ada in Denmark
jb@ada-dk.org

Exhibition Chair

Peter Dencker
ETAS GmbH
peter.dencker@etas.com

Publicity Chair

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

Local Chair

Raúl Rojas
FU Berlin
Raul.Rojas@fu-berlin.de

Local Organizer

Christine Harms
christine.harms@ccha.de

In cooperation (requests pending) with
ACM SIGAda, SIGBED, SIGPLAN

General Information

The 18th International Conference on Reliable Software Technologies – Ada-Europe 2013 will take place in Berlin, Germany. Following its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical and industrial sessions, along with parallel tutorials and workshops on Monday and Friday.

Schedule

3 December 2012	Submission of regular papers, tutorial and workshop proposals
14 January 2013	Submission of industrial presentation proposals
11 February 2013	Notification of acceptance to all authors
10 March 2013	Camera-ready version of regular papers required
11 May 2013	Industrial presentations, tutorial and workshop material required

Topics

The conference has successfully established itself as an international forum for providers, practitioners and researchers into reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies.
To mark the completion of the **Ada 2012** standard revision process, contributions are sought that discuss experiences with the revised language.

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

- **Multicore Programming**: Reliable Parallel Software, Scheduling on Multi-Core Systems, Compositional Parallelism Models, Performance Modelling, Deterministic Debugging.
- **Real-Time and Embedded Systems**: Real-Time Software, Architecture Modelling, HW/SW Co-Design, Reliability and Performance Analysis.
- **Theory and Practice of High-Integrity Systems**: Distribution, Fault Tolerance, Security, Reliability, Trust and Safety, Languages Vulnerabilities.
- **Software Architectures**: Design Patterns, Frameworks, Architecture-Centered Development, Component and Class Libraries, Component-based Design and Development.
- **Methods and Techniques for Software Development and Maintenance**: Requirements Engineering, Object-Oriented Technologies, Model-driven Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- **Enabling Technologies**: Compilers, Support Tools (Analysis, Code/Document Generation, Profiling), Run-time Systems, Distributed Systems, Ada and other Languages for Reliable Systems.
- **Software Quality:** Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Energy, Games and Serious Games, etc.
- **Experience Reports in Reliable System Development**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **Experiences with Ada and its Future**: New Language Features, Implementation and Use Issues; Positioning in the Market and in Education; where should Ada stand in the Software Engineering Curriculum; Lessons Learned on Ada Education and Training Activities with bearing on any of the conference topics.

Program Committee

Ted Baker, US National Science Foundation, USA
Johann Blieberger, Technische Universität Wien, Austria
Bernd Burgstaller, Yonsei University, Korea
Alan Burns, University of York, UK
Rod Chapman, Altran Praxis Ltd, UK
Dirk Craeynest, Ada-Belgium & KU Leuven, Belgium
Juan A. de la Puente, Universidad Politécnica de Madrid, Spain
Franco Gasperoni, AdaCore, France
Michael González Harbour, Universidad de Cantabria, Spain
Xavier Grave, Centre National de la Recherche, France
Christoph Grein, Ada Germany, Germany
J. Javier Gutiérrez, Universidad de Cantabria, Spain
Peter Hermann, Universität Stuttgart, Germany
Jérôme Hugues, ISAE Toulouse, France
Pascal Leroy, Google, Switzerland
Albert Llemosí, Universitat de les Illes Balears, Spain
Kristina Lundqvist, Mälardalen University, Sweden
Franco Mazzanti, ISTI-CNR Pisa, Italy
John McCormick, University of Northern Iowa, USA
Stephen Michell, Maurya Software, Canada
Luís Miguel Pinho, CISTER Research Centre/ISEP, Portugal
Jürgen Mottok, Regensburg University of Applied Sciences, Germany
Manfred Nagl, RWTH Aachen University, Germany
Laurent Pautet, Telecom ParisTech, France
Jorge Real, Universitat Politécnica de València, Spain
Jean-Pierre Rosen, Adalog, France
José Ruiz, AdaCore, France
Ed Schonberg, AdaCore, USA
Tucker Taft, AdaCore, USA
Theodor Tempelmeier, Univ. of Applied Sciences Rosenheim, Germany
Elena Troubitsyna, Åbo Akademi University, Finland
Tullio Vardanega, Università di Padova, Italy
Juan Zamorano, Universidad Politécnica de Madrid, Spain

Industrial Committee

Jørgen Bundgaard, Rambøll Danmark, Denmark
Jacob Sparre Andersen, JSA, Denmark
Jamie Ayre, AdaCore, France
Ian Broster, Rapita Systems, UK
Rod Chapman, Altran Praxis Ltd, UK
Dirk Craeynest, Ada-Belgium & KU Leuven, Belgium
Michael Friess, AdaCore, France
Ismael Lafoz, Airbus Military, Spain
Ahlan Marriott, White-Elephant GmbH, Switzerland
Steen Ulrik Palm, Terma, Denmark
Paolo Panaroni, Intecs, Italy
Paul Parkinson, Wind River, UK
Ana Isabel Rodríguez, GMV, Spain
Jean-Pierre Rosen, Adalog, France
Alok Srivastava, TASC Inc, USA
Claus Stellwag, Elektrobit AG, Germany
Jean-Loup Terraillon, European Space Agency, The Netherlands
Rod White, MBDA, UK

Call for Regular Papers

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall be in English, complete and not exceeding 14 LNCS-style pages in length. Authors should submit their work via the EasyChair conference system (*http://www.easychair.org/conferences/?conf=ae13*). The format for submission is solely PDF. For any remaining questions, please contact a *Program Co-Chair*.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the start of the conference. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by March 10, 2013. For format and style guidelines authors should refer to the following URL: *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The conference is ranked class A in the CORE ranking, is among the top quarter of CiteSeerX Venue Impact Factor, and listed in DBLP, SCOPUS and the Web of Science Conference Proceedings Citation index, among others.

Awards

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

Call for Industrial Presentations

The conference also seeks industrial presentations which deliver value and insight, but may not fit the selection process for regular papers. Authors of industrial presentations are invited to submit an overview (at least 1 full page in length) of the proposed presentation by January 14, 2013, via the EasyChair conference system (*http://www.easychair.org/conferences/?conf=ae13*). The *Industrial Committee* will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by May 13, 2013, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the *Industrial Chair* directly.

Call for Tutorials

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

Call for Workshops

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

Call for Exhibitors

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

Grant for Reduced Student Fees

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