Safe and Secure Software

An Invitation to Ada 2005

Foreward / Contents / Introduction / Bibliography

Courtesy of AdaCore
The GNAT Pro Company

John Barnes

Foreword

The aim of this booklet is to show how the study of Ada in general and Ada 2005 in particular, is helpful to everyone designing safe and secure software regardless of the programming language in which the software is eventually written. After all, successful implementers of safe and secure software write in the spirit of Ada in any language!

Thank you John for showing this throughout your papers, rationales, books, and this booklet.

AdaCore dedicates this booklet to all the designers and implementers of safe and secure software.

Contents

Introduction

The aim of this booklet is to show how Ada 2005 addresses the needs of designers and implementers of safe and secure software. The discussion will also show that those aspects of Ada that make it ideal for safety-critical and security-critical application areas will also simplify the development of robust and reliable software in many other areas.

The world is becoming more and more concerned about both safety and security. Moreover, software now pervades all aspects of the workings of society. Accordingly, it is important that software which is concerned with systems for which safety or security are a major concern should be safe and secure.

There has been a long tradition of concern for safety going back to the development of railroad signaling and more recently with aviation. Vital software systems such as those that control aircraft navigation and landing have to meet well established certification and validation criteria.

More recently there has been growing concern with security in systems such as banking and communications generally. This has been heightened with concern for the activities of terrorists.

Safety and security are intertwined through communication. An interesting characterization of the difference is

- safety – the software must not harm the world,
- security – the world must not harm the software.

So a safety-critical system is one in which the program must be *correct*, otherwise it might wrongly change some external device such as an aircraft flap or a railroad signal, with serious real-world consequences.

And a security-critical system is one in which it must not be possible for some incorrect or malicious input from the outside to violate the integrity of the system, for example by corrupting a password checking mechanism and stealing social security information.

The key to guarding against both problems is that the software must be correct in the aspects affecting the system's integrity. And by correct we mean that it meets its specification. Of course if the specification is incomplete or itself incorrect then the system will be vulnerable. Capturing requirements correctly is a hard problem and is the focus of much attention from the lean software development community.

One of the trends of the second half of the twentieth century was a universal concern with freedom. But there are two aspects of freedom. The ability of the

individual to do whatever they want conflicts with the right to be protected from the actions of others. Maybe A would like the freedom to smoke in a pub whereas B wants freedom from smoke in a pub. Concern with health in this example is changing the balance between these freedoms. Maybe the twenty-first century will see further shifts from "freedom to" to "freedom from".

In terms of software, the languages Ada and C have very different attitudes to freedom. Ada introduces restrictions and checks, with the goal of providing freedom from errors. On the other hand C gives the programmer more freedom, making it easier to make errors.

One of the historical guidelines in C was "trust the programmer". This would be fine were it not for the fact that programmers, like all humans, are frail and fallible beings. Experience shows that whatever techniques are used it is hard to write "correct" software. It is good advice therefore to use tools that can help by finding bugs and preventing bugs. Ada was specifically designed to help in this respect. There have been three versions of Ada – Ada 83, Ada 95 and now Ada 2005.

The purpose of this booklet is to illustrate the ways in which Ada 2005 can help in the construction of reliable software, by illustrating some aspects of its features. It is hoped that it will be of interest to programmers and managers at all levels.

It must be stressed that the discussion is not complete. Each chapter selects a particular topic under the banner of *Safe X* where *Safe* is just a brief token to designate both safety and security. For the most critical software, use of the related SPARK language appears to be very beneficial, and this is outlined in Chapter 11.

A topic with which Ada has much synergy is lean software development – there is not enough space in this booklet to expand on this concept but the reader is encouraged to explore its good ideas elsewhere.

As the twenty-first century progresses we will see software becoming even more pervasive. It would be nice to think that software in automobiles for example was developed with the same care as that in airplanes. But that is not so. My wife recently had an experience where her car displayed two warning icons. One said "stop at once", the other said "drive immediately to your dealer". Another anecdotal motor story is that of a driver attempting to select channel 5 on the radio, only to see the car change into 5th gear! Luckily he did not try Replay.

For a fuller description of Ada 2005, SPARK, and lean software development and papers on related topics please consult the bibliography.

Bibliography

The following two books are comprehensive descriptions of Ada 2005 and SPARK respectively. Both contain CDs with appropriate supporting material.

John Barnes. *Programming in Ada 2005*. Addison-Wesley (2006).

John Barnes with Praxis Critical Systems. *High Integrity Software – The SPARK approach to Safety and Security.* Addison-Wesley (2003).

The following award-winning book is a good introduction to lean software development.

Peter Middleton, James Sutton. *Lean Software Strategies: Proven Techniques for Managers and Developers*. Productivity Press (2005).

The following websites provide access to much useful information.

www.adacore.com – for AdaCore and its products.

www.ada-europe.org – for Ada-Europe, conferences and journal.

www.adaic.org – for the Ada Information Clearinghouse.

www.sparkada.com – for SPARK.

The following further documents and books are referenced in the text.

[1] *Software Considerations in Airborne Systems and Equipment Certification,* DO-178B/ED-12B, RTCA_EUROCAE. (December 1992).

[2] Cyrille Comar and Pat Rogers. *On Dynamic Plug-in Loading with Ada 95 and Ada 2005*. AdaCore (2005). http://www.adacore.com/.

[3] ISO/IEC TR 24718:2004. Guide for the use of the Ada Ravenscar profile in high integrity systems. (2004).

[4] Alan Burns and Andy Wellings. *Concurrent and Real-Time programming in Ada 2005*. Cambridge University Press (2006).

[5] Janet Barnes, Rod Chapman, Randy Johnson, James Widmaier, David Cooper and Bill Everett. *Engineering the Tokeneer Enclave Protection Software.* Published in ISSSE 06, the proceedings of the 1st IEEE International Symposium on Secure Software Engineering. IEEE (March 2006). Also available from www.sparkada.com.

North American Headquarters
104 Fifth Avenue, 15th floor
New York, NY 10011-6901, USA
tel +1 212 620 7300
fax +1 212 807 0162
sales@adacore.com
www.adacore.com

European Headquarters
46 rue d'Amsterdam
75009 Paris, France
tel +33 1 49 70 67 16
fax +33 1 49 70 05 52
sales@adacore.com
www.adacore.com