23rd International Conference on

RELIABLE SOFTWARE TECHNOLOGIES

Ada-Europe 2018

18-22 June 2018, Lisbon, Portugal

PRESENTATION ABSTRACTS

In cooperation with

Ada Resource Association

TABLE OF CONTENTS

Table of Contents 2

Part 1: Presentations in Regular Sessions 3

The IRONSIDES Project: Final Report 5
Concurrent Reactive Objects in Rust – Secure by Construction 7
Alire: a Library Repository Manager for the Open Source Ada Ecosystem 9
Real-Time Ada Applications on Android 11

Part 2: Presentations in Industrial Sessions 13

Managing the Endianness of Software Building Blocks with GNAT Ada Pragmas: a Case Study 15
Using Ada in Non-Ada Systems 17
Easy Ada Tooling with Libadalang 19
Ariane 6 Flight Software Designed for a Simpler Validation 21
I3DS - A Modular Sensor Suite for Space Robotics 23
Multi-Concern Dependability-Centered Assurance for Space Systems via ConcertoFLA 25
Applying Formal Timing Analysis to Satellite Software 27
Multicore Timing Analysis for Safety-Critical Software 29
KhronoSim: Simulation and Testing of Real-Time Critical Cyber-Physical Systems 31
C Guidelines Compliance and Deviations (the MISRA and CERT Cases) 33
Agile in Safety Critical Projects 35
AGILE-R: Agile Software Development for Railways 37

Organization 40

Program Committee 40

Conference Sponsors 40

PART 1: PRESENTATIONS IN REGULAR SESSIONS

The IRONSIDES Project: Final Report

In a project intended to improve the security of internet software, the authors developed IRONSIDES: A DNS server written in Ada/SPARK. Our long-term goals were a) to show that a fully functional component of the internet software suite could be written with provably better security properties than existing alternatives, b) to show that it could be done within the relatively modest resources available for a research project at an undergraduate university, c) to determine the suitability of Ada/SPARK for such a project, and d) to compare the performance of the resulting software to existing alternatives and determine to what extent, if any, the addition of provable security properties affects performance. We report our conclusions from this multi-year project.

The IRONSIDES Project

The authors believed many of the security problems with DNS servers, web servers, and other internet software could be avoided with the use of better programming tools, such as the use of different programming languages and formal methods. They chose Ada and SPARK as an appropriate development environment to implement a provably secure DNS server from the ground up,

The SPARK language and toolset from Altran UK is used in the creation of software systems with provable correctness and security properties. SPARK is a subset of Ada, augmented with special annotations. These annotations appear as ordinary comments to Ada compilers, but are visible to SPARK's pre-processing tools used to validate software. SPARK is a mature technology and has been used on several projects, including an open-source OS kernel provably free from runtime errors, the British Air Traffic Control System, and multi-level security workstations. Accordingly, given our prior institutional experience with Ada, we chose SPARK and Ada as the platform for constructing DNS software that would not be subject to most of the vulnerabilities that afflict DNS implementations currently deployed around the world.

The SPARK toolset generates verification conditions (VC's) that it then attempts to verify. VCs include assertions that variables always remain in type, array bounds are never exceeded (a common source for buffer overflow vulnerabilities), pre- and post- conditions are always met, and so forth. When a VC has been proved by SPARK, it is said to be discharged.

The project contained 3 milestones:

Milestone 1: An authoritative server on Ubuntu

The first IRONSIDES milestone was achieved with the successful construction of an authoritative server, tested against BIND on Ubuntu.

We were pleasantly pleased to discover that the authoritative IRONSIDES DNS server performed significantly better than BIND under Linux.

Milestone 2: An authoritative server on Windows

The next milestone was porting IRONSIDES to Windows,, and testing it against both WinDNS and BIND/ The test bed was similar, except the virtual machine used ran Windows Server 2008. Performance results are shown below:

We fully expected IRONSIDES to perform better than BIND, but were surprised to find it outperformed Windows DNS on its own native OS by 7%.

Milestone 3: A recursive server and detailed performance comparisons

Recursive servers are more complex than authoritative ones, requiring more sophisticated data structures, cache management, and tasking. Building on our experience with the authoritative version, we next added recursive query functionality to IRONSIDES.

Once we had produced a validated recursive server, we were ready to do a detailed performance comparison with a variety of both open-source and proprietary DNS servers. We expanded the test bed to include a virtual machine running each server/OS combination, a VM running the Resperf performance analyzer, and a VM running the network simulator INETSIM.

When we ran the server in authoritative mode under Ubuntu, IRONSIDES continued to outperform BIND and others, although the gap had narrowed from about 3x to about 2x:

Under Windows, however, WinDNS now performed slightly better, perhaps due to improvements in later releases or the increased complexity of IRONSIDES required to support recursive queries:

Up to 1500 queries per second, the performance of all the servers was essentially indistinguishable. At higher values, IRONSIDES, DNSMASQ and DJBDNS dropped off fairly rapidly. Surprisingly, under Windows, BIND also did the best.

On the other hand, in terms of queries lost, WinDNS and IRONSIDES performed best:

IRONSIDES had the second lowest latency for Unix DNS servers, but the longest latency for Windows servers. We believe this is due to latency being extremely important to Microsoft, and to IRONSIDES policy of trying to handle every query it can (BIND, by contrast, drops queries if it is too busy):

Insights from experience

The results from the authoritative server design process gave our first hints that performance did not need to be sacrificed to improve security. In fact, there were clear examples in which the use of formal methods actually improved performance. For example, data flow analysis identified redundant or ineffective statements that in turn permit the removal of inefficient code. Code that has been proven exception-free no longer requires runtime bounds checking, so that can be eliminated as well.

We also learned, however, that there were cases were total reliance on formal methods and proof negatively impacted performance. Because SPARK requires all data structures to be statically allocated, data structures must be sized at the upper limits of expected use. Explicit initialization of such structures, while required for validation, is inefficient and wasteful. In those rare cases, we explicitly told the tools to relax that requirement. This improved IRONSIDES performance by almost 30%. Thus we believe allowing users to override formal proof requirements when appropriate is an important feature that formal methods tools should always support.

It is crucial to always remember the role of the compiler. Despite our confidence in the tools to help us produce crash-proof software, we found one combination of operating system, compiler and optimization level where a fully validated version of IRONSIDES crashed with an exception. This was due to a code generation error in the version of GNAT shipped with Ubuntu, long since corrected. Still, until formal methods have progressed sufficiently to the point where they can prove the correctness of compilers for a given target architecture and OS, programmers should continue to exercise healthy skepticism when compiling and testing verified software.

Our experience with the tools produced results we would describe as both impressive and humbling. Despite both of us having computer science PhD's, over 50 years of combined industry and academic experience, and an extensive knowledge of programming languages and software engineering practice, the tools still caught boundary conditions and potential problems that in principle we could have found but did not. This is the whole point of using formal systems, but the experience is nonetheless humbling. Perhaps it will become less so as formal methods and proof tools become a standard part of the software engineering process.

IRONSIDES has numerous provable security properties absent from all the other servers tested, including BIND and WinDNS. These include:

1) No classic buffer overflow
2) No incorrect calculation of buffer size
3) No improper initialization
4) No ineffective statements
5) No integer overflow/wraparound
6) No information leakage
7) All input validated
8) No allocation w/o limits (no resource exhaustion)
9) No improper array indexing
10) No null pointer dereferencing
11) No expired pointer dereferencing (use after free)
12) No type confusion
13) No race conditions
14) No incorrect conversions
15) No uncontrolled format strings
16) All loops guaranteed to terminate

With all these advantages, we were pleased to discover that IRONSIDES also performs comparably to servers with security problems, including the industry standards of BIND and WinDNS. IRONSIDES offers comparable performance at nominal loads, trailing off only under maximal loading. This is particularly significant considering each server's respective development costs. BIND is produced with an industrial consortium. WinDNS is bundled with the flagship product of a multibillion dollar software company. IRONSIDES was written by the equivalent of a little more than one professor at an undergraduate university with near full-time teaching duties.

4 Conclusions and future work

The success of IRONSIDES indicates that formal methods can be used both improve the security properties of software without incurring significant performance penalties, and in some cases can actually improve performance. This was done in an environment with significantly fewer resources available than comparable products.

We hope this work will be further extended to apply formal methods and performance analysis outside the DNS domain, in the hopes of continued confirmation that internet software can be made provably more secure without significant sacrifices in performance. Web servers, for example, suffer from similar security problems for similar reasons. ICS and SCADA systems are currently attractive targets for hacking, and formal methods have been used to improve their security, but the effect of formal methods on performance in this domain remains unknown. These are the subject of current work at the Academy Center for Cyberspace Research.

Acknowledgments

This work was supported in part by the US Air Force Office of Scientific Research under grant #1220961, the US Department of Defense Advanced Research Projects Agency, and the Academy Center for Cyberspace Research.

Concurrent Reactive Objects in Rust Secure by Construction

Marcus Lindner, Jorge Aparicio, and Per Lindgren

Luleå University of Technology

Email:{marcus.lindner, jorge.aparicius, per.lindgren}@ltu.se

Abstract

Commonly, embedded software autonomously operates with requirements on safety, reliability, and security, besides constraints (e.g., memory, CPU, energy resources) set by the environment (i.e., target platform) and other functional and extra-functional properties of the application at hand. In general, embedded systems of the IoT-era are realized by taking the outset of a reactive model implemented in C/C++ either as a bare metal interrupt driven application or through the support of some threading library. Meeting the aforementioned requirements is at a large up to the programmer with little or no support for verification. Central to correctness is the management of memory resources, with problems spanning from array indexing, dangling pointers, etc. all the way to race conditions and deadlocks in the concurrent setting.

In this paper, we take the outset from prior work on Concurrent Reactive Objects (CROs) [3] with a heritage to the Timber language ([4]) and the Real-time For the Masses (RTFM) set of experimental languages and tools. Whereas Timber provides a high level modelling and implementation approach offering state protection in the concurrent setting, the dynamic memory model requires automatic management which precludes the deployment to light weight targets.

The motivation is clear, we want to provide a programming model ensuring memory safety in a concurrent setting along with a concurrency model amenable to static analysis. However, developing yet another fully fledged language with accompanying ecosystem is questionable when taking the amount of work into consideration[1]. Instead, we seek to leverage on ongoing community efforts around programming languages and ecosystems.

Among recent developments, the Rust language stands out with a memory model providing compile time memory safety, monomorphization, and a tight coupling to LLVM (achieving zero-cost abstractions through link time optimization). Sidestepping the compiler is explicit (`unsafe`) and can be rejected in user

[1] An observation here is that the design of any memory safe language would need to take memory aliasing into account, a property directly given by the Rust language.

code, thus allowing for fearless programming to the end of the memory safety and other properties within reach of the Rust compiler. In the context of embedded development, Rust applications on bare metal targets have already been shown possible [2, 1].

In this paper, we further explore the Rust memory model and the CRC framework towards systems being secure by construction. In particular, we show the following properties.

- Granted permissions can be freely delegated without any risk of leakage outside of the intended set of components. Key here is the static CRC topology, where communication paths are known at compile time together with the Rust language borrowing semantics.
- Permissions are guaranteed to be authentic (can neither be manipulated nor faked). Key here is the underlying module system and (type) scoping together with the memory safety provided by the Rust language (in effect, preventing any intentional or accidental memory corruption leading up to a unauthentic permission).
- Permissions are guaranteed to be temporal (can never outlive the granting authority). Key here is the concept of lifetimes brought by the Rust language and enforced by the compiler.

In conclusion, we believe and argue that these properties offer the fundamental primitives for building secure-by-construction applications and demonstrate its feasibility on a small case study, a wireless autonomous system based on a ARM Cortex M3 target.

References

[1] E. Holk, M. Pathirage, A. Chauhan, A. Lumsdaine, and N. D. Matsakis. Gpu programming in rust: Implementing high-level abstractions in a systems-level language. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 315–324, May 2013.

[2] A. Levy, B. Campbell, B. Ghena, P. Pannuto, P. Dutta, and P. Levis. The case for writing a kernel in rust. In *Proceedings of the 8th Asia-Pacific Workshop on Systems*, APSys '17, pages 1:1–1:7, New York, NY, USA, 2017. ACM.

[3] J. Nordlander, M. P. Jones, M. Carlsson, R. B. Kieburtz, and A. Black. Reactive objects. In *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002*, pages 155–158, 2002.

[4] The Timber Language, webpage. http://www.timber-lang.org, last accessed 2017-09-16.

Alire: a library repository manager for the open source Ada ecosystem

Alejandro R. Mosteo[1,2]

[1] Instituto de Investigación en Ingeniería de Aragón (I3A)
amosteo@unizar.es,
Mariano Esquillor s/n, 50018, Zaragoza, Spain
[2] Centro Universitario de la Defensa de Zaragoza (CUD)
Ctra. de Huesca s/n, 50090, Zaragoza, Spain

Abstract. Open source movements are main players in today's software landscape. Communities spring around programming languages, providing compilers, tooling and, chiefly, libraries built with these languages. Operating systems providers and distributions often package but the most significant or mature libraries so, usually, language communities develop their own cross-platform software management tools. Examples abound with languages such as Python, OCaml, Rust, Haskell, etc.
The Ada community has been an exception to date, perhaps due to its smaller open source community. This work presents a working prototype currently tailored to the Ada compiler available to open source enthusiasts, GNAT. This tool is designed from two main principles: zero-cost infrastructure and a pure Ada implementation. The semantic versioning paradigm is used for dependency resolution, with Ada specification files describing project releases and dependencies.

Keywords: Library Management, Dependency resolution, Open Source, Ada 2012

1 Introduction

In nowadays rapidly evolving technological landscape, reuse of code is critical to adapt to new technologies, avoid past errors, stay on top of vulnerabilities, and foster collaboration. This collaborative spirit is prominently seen in the publishing of software under more or less permissive licenses [3], but free of charge.

In some cases, like Rust [2], a tool for the distribution of libraries is an integral effort of the team developing the language. Such a tool increases visibility and availability of libraries, and also deals with the resolution of dependencies.

The Ada language, perhaps because of its ties to closed development and today's considered niche place in the language landscape [1], has not seen such a tool appear yet. This work presents what could be a first step in this direction, presenting the Alire project and its command-line `alr` tool. This tool tries to appeal to the Ada programmer by using compilable Ada code to describe releases and its dependencies, thus avoiding the need to learn new formats.

2 Presentation aim and technical overview

During this presentation, the interested participant will learn about the basics of operation of the `alr` tool and the subjacent design and supporting infrastructure. The current status of the project and foreseeable evolution will be discussed. The Alire project enables the open source Ada developer to:

- Package a library or application for dissemination to the Ada community.
- Reuse other Ada projects with ease.
- Update third-party dependencies when new features or fixes are issued.

Some technical highlights of the presented solution are:

- It is designed for user-space, avoiding platform configuration issues.
- Each working project has sandboxed dependencies.
- Does not depend on complex or paid services.
- An indexed release is described using Ada code verified by compiling it.
- Developers are minimally impacted for integration into Alire: a GPR project file is the only current requirement.

3 Conclusion

The Alire project proposes an experimental Ada tool that facilitates the dissemination and reuse of Ada projects. This is achieved by indexing semantic versions of source releases in public repositories, which in turn enables the possibility of dependency resolution and easy upgrades. The design is based around metadata files written in Ada.

Alire is available under an open source license to interested parties[3].

Acknowledgements

Work supported by projects ROBOCHALLENGE (DPI2016-76676-R-AEI/FEDER-UE), ESTER (CUD2017-00), and SIVIDRA (UZCUD2017-TEC-06).

References

1. Hamilton, D., Pape, P.: 20 years after the mandate. CrossTalk p. 15 (2017)
2. Matsakis, N.D., Klock II, F.S.: The Rust language. SIGAda Ada Letters 34(3), 103–104 (2014)
3. Peterson, C.: How I coined the term 'open source', `https://opensource.com/article/18/2/coining-term-open-source-software`

[3] `https://bitbucket.org/aleteolabs/alire`

Real-time Ada applications on Android

Alejandro Pérez Ruiz, Mario Aldea Rivas, and Michael González Harbour

Software Engineering and Real-Time Group
Universidad de Cantabria, 39005 - Santander, SPAIN
{perezruiza, aldeam, mgh}@unican.es
http://www.istr.unican.es/

Extended abstract[1]

Android is the most extended operating system in the field of smartphones and as such it is subject to permanent evolution with new features being implemented continuously. It has been adapted to many kinds of devices. Because its popularity, there is a great interest in using it in areas where mobile devices have an increasing role, including industrial control, medical environments or automotive, in which real-time requirements are common.

The Ada language is a mature and effective technology for developing concurrent programs with timing requirements, given its native support for these features. Its emphasis on reliability makes it suitable for developing safety-critical applications. The main objective of this work is to build a platform that allows us to develop real-time Ada applications on top of the Android operating system.

As we have already described in a previous work [1], given that Android is based on the Linux kernel it offers mechanisms that allow us to isolate a core in a multi-core processor, which can be used to execute native applications with timing requirements, while avoiding the interactions with non-real-time applications.

Android consists of different layers of software components. The lower layer corresponds to the Linux kernel, which provides the basic operations of the system, such as memory management, processes and drivers. Over the kernel layer lays a set of native libraries that are written in C or C++ and are compiled for a specific hardware architecture. These libraries include a modification of the standard C library (glibc) that it is called Bionic in Android. This library is designed specifically for Android but we have detected that some essential functions for real-time purposes, such as those related to mutex protocols, are not implemented. The first consideration to solve the limitations of the Bionic library could be to modify its code to support the unimplemented functions. This would involve a great effort and constant adaptation to future new versions of the library. Therefore, we have decided to opt for a more portable and easy solution that consists in replacing Bionic with the traditional glibc library.

If we built the Ada compiler for an Android target it would be using the Bionic library, and that is why we choose a Linux target, which uses glibc. Anyhow, the code

1 This work has been funded in part by the Graduate Grant Program of the University of Cantabria and the Spanish Government under grant number TIN2014-56158-C4-2-P (M2C2).

generated for both targets is binary compatible, since Android is based on the Linux kernel. As a consequence, our platform is built from a GNAT FSF cross-compiler for an x86_64/Linux environment (host) that allows us to compile Ada programs for an ARM/Linux device (target).

To ensure that the replacement of the Bionic library does not cause failures, we have adapted an existing script to run the Ada Conformity Assessment Test Suite (ACATS) [2] on an Android device and we have verified that all the tests pass successfully with the generated compiler and libraries.

Moreover, in this work we have studied the available mechanisms in Android to share data between the isolated real-time Ada applications and other applications executing in other cores of the same system.

To sum up, we have shown that it is possible to build a platform for executing soft real-time Ada applications on the Android operating system by combining the solution proposed in our previous work [1], which isolates a core in the system to dedicate it to real-time applications, with the compilation and execution of Ada programs on Android.

References

1. Alejandro Pérez Ruiz, Mario Aldea Rivas and Michael González Harbour: “CPU Isolation on the Android OS for running Real-Time Applications”: Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), 2015.
2. Brukardt, R.L: Ada Conformity Assessment Test Suite (ACATS), http://www.ada-auth.org/acats.html

PART 2: PRESENTATIONS IN INDUSTRIAL SESSIONS

MANAGING THE ENDIANNESS OF SOFTWARE BUILDING BLOCKS WITH GNAT ADA PRAGMAS: A CASE STUDY

Patricia Lopez Cueva, Marco Panunzio

Thales Alenia Space

Contact: Marco Panunzio
marco.panunzio@thalesaleniaspace.com
5 Allées de Gabians – BP 99
06156 Cannes La Bocca Cedex - France

1. CONTEXT

In the past 20 years, the European space industry has used hardware platforms mostly based on the SPARC architecture (ERC32, LEON2, LEON3 processors) and occasionally on Power PC.
Both architectures are big-endian.
A recent trend for next generation platforms is to adopt ARM-based Systems on Chips (SoC), such as DAHLIA [1], which are natively little-endian.
The difference in endianness makes it more difficult to reuse and adapt existing software, which is typically inherited with extension and accommodation across subsequent missions.

Furthermore, it is expected that there would be a gradual migration of architecture, which would require the development of certain new missions still with traditional hardware architectures and some others with new ARM based architectures.

This topic is then particularly relevant for embedded software where data storage and memory layout are among the prevalent concerns (typically the lower layers of the architecture).

2. ANALYSIS AND POSSIBLE SOLUTIONS

Addressing the endianness of software is typically addressed with 2 different strategies:

1) by developing different versions of the software for each endianness, which could be optimized for a given architecture and memory layout. However, such solution is much more expensive in terms of effort for development, configuration management, V&V and maintenance.

2) by having a single version of the software, and managing endianness by mechanisms such as the use of preprocessing macros to adapt the code for a chosen endianness.
The latter solution is typically used in C / C++ implementations.

The Ada GNAT compiler offers a rather elegant solution to this problem, providing compiler specific pragmas to define the endianness of Ada records, i.e. the *Bit_Order* and the *Scalar_Storage_Order* pragmas. In theory, these pragmas allow to implement an almost transparent solution for the software as endianness consideration appears only at data type declaration and does not impact functional / implementation code.

3. AN APPLICATION CASE STUDY

The mechanisms offered by the GNAT compiler could give an interesting means to facilitate the migration of existing building blocks in Ada to architectures with different endianness and to permit to manage a single software application implementation maintained on both architectures.

In order to take advantage of these mechanisms, and to derive qualitative and quantitative conclusions on it, we have performed a prototype modification of a

sizeable software building block in our architecture, the PUS library.

The PUS library is an implementation of the ECSS PUS standard [2].
The PUS library is the core of the Thales Alenia Space software execution platform, and manages the major protocol exchanges for Telecommands and Telemetry (TM / TC) between the satellite and an external systems (typically a ground control station).
It has a considerable flight heritage, as it has been used (with subsequent improvements) in all Thales Alenia Space platform software of the past 10 years: Globalstar 2, O3B, Iridium Next, Sentinel 3, Exomars-TGO and EDM, Spacebus NEO, as well as recent optical observation satellites and telecom payloads.
Furthermore, the same PUS library is used by other European software integrators on both satellite platform software or instruments / payloads.

The PUS library is one of the major components of the on-board software (~ 30000 Ada SLOC) and is quite impacted by endianness change.

The size and importance of this software building block made it a quite interesting case study for testing in practice the mechanisms described before.

In the scope of this presentation, we will illustrate some practical examples of how we had to restructure the original software code and what were the concrete - foreseen (or unforeseen) - consequences of the application of the GNAT pragma directives in the code.

The prototype modifications have been tested on a setup consisting of a test bed with an Intel host computer, with big-endian communication messages (as per the CCSDS / PUS standard) transmitted to a little-endian Zedboard target hardware (including a dual-core ARM A9 processor).

We will then report on the outcome of the activity, the problems that were identified, and possible solutions or mitigations.

Finally, we will provide some conclusions in terms of usability, impact on development and test of implementation of using this approach to endianness, as well as proposing some recommendations.

4. REFERENCES

1. DAHLIA: Deep sub-micron microprocessor for spAce rad-Hard appLIcation Asic. H2020 Project ID: 730011 http://dahlia-h2020.eu/

2. ECSS-E-70-41A: Software Engineering - Ground systems and operations - Telemetry and telecommand packet utilization

Using Ada in non-Ada microprocessor based Systems.

Presented by Ahlan Marriott and Urs Maurer, White Elephant GmbH, Switzerland

This industrial presentation is an experience report of our use of Ada packages within existing non-Ada embedded microprocessor systems.

We have a large amount of well-established code that uses our own proprietary multitasking run-time. Management is unlikely to sanction the conversion of this code base into Ada because the risk of introducing errors far out ways any perceived benefits of coding in Ada.

However this is not to say that new features or features that have to be substantially modified couldn't be written in Ada.

The presentation will describe our experience integrating Zero Footprint Ada into such an environment.

The existing code is primarily written in ISO 10514 Modula-2, which is compiled into the ARM target machine code by first translating it into C and then using the GCC tool chain to compile and link this C for a wide range of target architectures.

Because our latest hardware will use an ARM based microprocessor and because GNAT now supports Ada for ARM microprocessors, we can, at least in theory, write Ada packages and use the same GCC that we use to compile C to compile and link these packages into an existing C based system.

This presentation is an experience report on using GCC version 6.3.1 as distributed by AdaCore as part of its GPL 2017 offering.

Ada is normally used in conjunction with its run-time. However the existing system into which we want to integrate Ada packages already has a run-time. On the assumption that it would be unwise to use more than one run-time and the impracticality of modify the existing system to use Ada's run-time we were consequently obliged to restrict our use of Ada to those features that do not require Ada's run-time. This is generally referred to as the Zero Footprint Profile.

Without Ada's run-time we cannot use:

- Tasks
- Protected objects
- Controlled types
- Dynamic storage allocation using *new*
- The *delay* statement;

Also, because our target microprocessor does not have floating point hardware, we restricted our packages to not use floating point and because our target has very little RAM, we also decided against implementing a secondary stack, which, if implemented, would require additional stack space for each and every task.

The presentation will describe how these restrictions were configured by use of complier switches, pragma restrictions and by the provision of certain RTS specification files.

Although our Ada is severely restricted, the use of even a reduced Ada still offers many advantages.
For example:

- Much improved and expressive syntax
- Named parameters
- Named fields in constructors
- Private types, functions and procedures
- Bit level specifications in representation clauses.

Representation clauses are extremely useful when interfacing to hardware and third party protocols where fields not represented by a complete byte are accessed in other programming languages by a combination of masks and shifts – typically an error prone endeavour which we are only too happy to delegate to Ada.

Our presentation will also cover topics such as:

- How Ada calls non-Ada code and how non-Ada code can call Ada.
- The need for wrapper routines.
- How global Ada variables are initialised.
- How the Ada packages are elaborated and how we establish the elaboration order.
- How we symbolically debug a system that is written in a mixture of Ada, Modula-2 and C.
- How Ada exceptions are handled.
- How to use the embedded assembler and how the assembler available to Ada differs from that used by C.
- Problems we had with interrupt routines.
- How, as an efficiency requirement, both Ada and C can be in-lined both within the same unit and between units by using the GCC's Link Time Optimization (LTO) feature but how this adversely affects symbolic debugging.

The presentation will also detail the problems we experienced with regard to type compatibility and name mangling.

Without a runtime, exceptions cannot be propagated outside the procedure that called them. They either have to be handled locally or caught by a last chance handler. We chose to not to provide a last chance handler and therefore we were forced to handle all exceptions locally or rewrite the code so that exceptions were impossible.

This decision caused a lot more work and problems than we initially expected and the presentation will describe some of these and the solutions we used.

Pre-elaboration was another major issue for us. Originally we thought that we could simply restrict ourselves to pre-elaborated and pure packages. However this turned out to be an unacceptable restriction and so we were obliged to integrate the elaboration of Ada packages into the existing mechanism whereby Modula-2 modules are elaborated.

The presentation will describe the mechanism we used and the subsequent need to enhance the Ada syntax, using our own pragma, to provide a means whereby the Ada code could inform our IDE that it uses a particular foreign unit and that this unit might need elaborating.

The presentation will end with a brief assessment of a feasibility study in which we converted the entire application code of two ARM specific products from Modula-2 into Ada.

Easy Ada Tooling with Libadalang

By Raphaël Amiard & Pierre-Marie de Rodat
Software engineers at AdaCore

Abstract

Libadalang is an Ada library that allows users to query information about Ada source code and to modify sources, both at syntactic and semantic levels. This talk will go over how to use Libadalang and what kind of tools can be created on top of it.

Extended abstract

A lot of developers consider that a language is only as good as the tooling that accompanies it. Ada has been conceived as a language pretty well amenable to tooling, yet the tooling offer besides AdaCore's is not very extensive, at least when compared to other languages like Java, despite the existence of the ASIS project (Ada Semantic Interface Specification).

One of Libadalang's aims is to help solve that by providing an easy way to build new Ada-aware tools. Libadalang is a library that allows the user to query information about Ada code.

Libadalang ships as an Ada library that exposes bindings to C and Python. It is designed to be ready for integration in every technology that accepts C extensions.

Libadalang's high level API

The Libadalang API, as described above, covers different levels of information. We will present how Libadalang exposes this information to the user at different levels, including:

- Syntactic information: query the token stream, the syntax tree, find syntax patterns, etc.
- Low level semantic information, such as: which declaration an identifier references, the type of expressions, …
- High level semantic information: all references to a declaration, derived types from a base type, overridden primitives, etc.
- Syntactic rewriting: Allow the users to modify the trees, and propagate the changes to the source.

Libadalang and ASIS

The Ada ecosystem already has a library specification to query information about source: ASIS. Libadalang differs from ASIS in several ways:

1. Works incrementally, so that it is usable to implement IDE features

2. Best effort approach with incorrect code:
    a. Can work on code only syntactically correct
    b. Can work with missing dependencies/etc
3. Evolving API. We plan to release several version of Libadalang, and while we will value backwards compatibility, we want to slowly evolve towards the best API possible to design code tools.

Tools based on Libadalang

We will present several tools based on Libadalang today:

- GNATpp, a pretty-printer for Ada
- GNATstub, a stub body generator for Ada
- GNATmetrics, a tool to compute various source-level metrics on Ada projects
- LAL-checkers, a framework to write simple source linters based on semantic information and control/data flow

Keywords

ada
tooling
library

Title
Ariane 6 flight software designed for a simpler validation

Authors
- Philippe GAST (philippe.gast@ariane.group)
  Institution : ArianeGroup
  City : Les Mureaux, France
- Cyrille PIERRE (cyrille.pierre@ ariane.group)
  Institution : ArianeGroup
  City : Les Mureaux, France

Abstract
Background
ArianeGroup is developing launchers and associated flight software since several decades. It is well known that a major part of the software development cost is related to its validation and qualification.
Then, it is essential to support these activities using ad hoc facilities; this is especially the case for validation facility which allows preparing the validation scenarios & debugging the flight software independently from the real time aspects.
The objective of the presentation is to present the logic applied by ArianeGroup to validate the Ariane 6 flight software and the link with the flight software design; the logic is based on using "host" and "target" validation facilities.

From flight software design to validation facilities
Depending of the flight software design, the representativeness of its behaviour on host environment can be low or high. For instance, on Ariane 5, mainly due to the fact that the flight software is mixing synchronous and asynchronous tasks, it is not possible to reproduce on host behaviour perfectly consistent with the one on target.
On Ariane 6 flight software, a full synchronous design approach has been applied as following
- Rate Monotonic Scheduling (RMS) is applied using Ravenscar profile to implement functions having different cyclic periods: frequencies of tasks are harmonic all together,
- Basic Cyclic Task (highest frequency and priority) is synchronized with avionics communication bus,
- Avionics exchanges based on LINO (Last In Next Out) mechanism avoid synchronization between flight software and avionics during an execution cycle.

Notice that acyclic events raised during the mission (stage release, engine ignition/stop, fault management…) are processed in a discrete way by the Basic Cyclic Task.

This design approach permits to have a host version of the flight software having a fully identical behaviour than the one on target (even if host version execution is not real time representative):
- The flight SW will processes on host or target the same avionics data in a dedicated cyclic execution cycle (execution of the software synchronised on avionic bus cyclic bus frame),
- No exchange of data between threads (no global variables) outside dedicated thread "rendez-vous" (which are synchronised with avionic bus frame,
- Independently from treatment execution duration: Scheduling of treatments in one cycle is the same and avionics measures / commands are received / prepared in the same cycle.

Considering this design approach, it is easy to build an emulated Software Validation Facility (SVF) which allows:

- developing and validating the flight software validation procedures,
- debugging flight software (behaviour identical with target behaviour) before formal validation campaign,
- debugging flight software most of the problems detected during flight software qualification : qualification execution traces can be replayed on emulated SVF

This approach permits to reduce the workload and to optimise the planning of the flight software validation.

The flight software design is strongly linked to the method which is applied to define the functional architecture of the system (called Functional Unit approach). This method is a reuse from Automated Transfer Vehicle project (orbital space vehicle); then the concepts which are presented here could be applied in various types of projects.

I3DS – a modular sensor suite for space robotics

Kristoffer Nyborg Gregertsen, PhD
SINTEF Digital, Department of Mathematics and Cybernetics

Space robotics has been identified as a key factor for improving the competitiveness of the European space industry. To fund research in this field through the H2020 programme, the European Commission has created the Space Robotics SRC (Strategic Research Cluster). Currently there are six ongoing Operational Grants (OG's) in the SRC, namely: OG1 for the development of a space robot control operating system (RCOS); OG2 for an autonomy framework; OG3 for a data fusion framework; OG4 for inspection sensor suite; OG5 for robotic payload interfaces; and OG6 for validation platforms and field tests. A call for follow-up operational grants to demonstrate the results is ongoing.

The I3DS (Integrated 3D Sensors) project carries out OG4, developing a modular sensor suite for space robotics. Thales Alenia Space is coordinator and partners are SINTEF, COSINE, University of Cranfield, PIAP, TERMA, and Hertz Systems. The project had its kick-off in November 2016, CDR in February 2018, and is as of April 2018 in the integration and testing phase. By the conference in June 2018 all software implementation, system integration and testing in the lab-bench setup is to be finished. Hence, the industrial presentation will contain fresh experiences from the system integration and testing at the SINTEF robotics and real-time systems lab, in addition to the system architecture and implementation. From July to November 2018 the system will be validated in two demonstrators at Thales Alenia Space lab facilities: one orbital rendezvous use-case for satellite inspection and servicing, and one planetary rover use-case for Mars exploration.

The author has the role of Interface Engineer, specifying and coordinating the sensor suite's interface with the other OG's, and leads the software development and system integration. The **reference architecture** of the I3DS project is given in AADL and defines the **system interface** of different **sensor classes** that are independent of the specific hardware implementation. This allows a mission-specific sensor suite to be composed in a modular way instead of new sensor suite software being developed for each mission. The different sensors communicate with the Onboard Computer (OBC) either directly or through an Instrument Control Unit (ICU) that acts as a gateway between the device specific protocol and the standard system interface. The sensor receives commands and sends measurement data in ASN.1 format following the design patterns of the TASTE framework used by OG1 ESROCOS. The autonomy (OG2 ERGO) and sensor fusion software (OG3 InFUSE) will run on the OBC.

The sensor suite developed in I3DS has the following sensors:

- A **high-resolution camera** that delivers a monochrome image stream.
- A **stereo camera** whose image streams can be processed into 3D point clouds.
- A **time of flight (TOF) camera** that captures depth images to generate 3D point clouds.
- A **LIDAR** that delivers distance measurements to be used to generate 3D point clouds.
- A **TIR camera** that delivers thermal imaging stream.
- A **star tracker** that finds the orientation and location of the vessel using the stars.
- A **radar** that is used for ranging measurements in rendezvous.
- An **IMU** that keeps track of the systems orientation and location.
- The **contact/tactile** and **force/torque** sensors for the final docking phase of a rendezvous.

In addition, a **projected pattern illumination** is to be used with the HR camera and a pre-processing algorithm to create 3D point clouds.

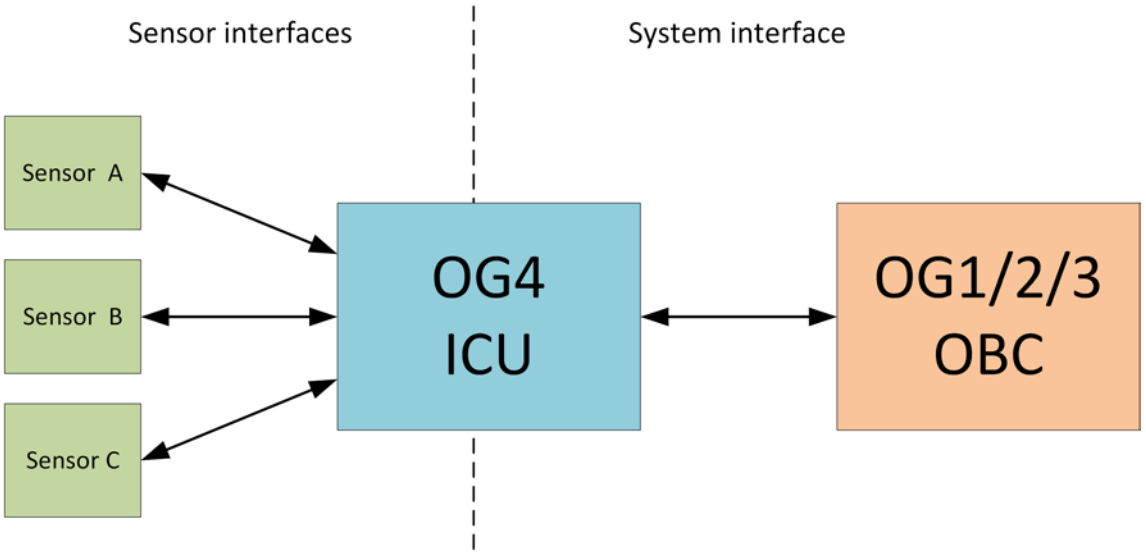


Figure 1. I3DS develops standardized system interfaces for the sensors, abstracting the device specific API and HW details.

In the I3DS instantiation of the reference architecture, the sensor suite is controlled and interfaced by an **ICU** implemented using the high-performance Xilinx Zynq UltraScale+ MPSoC in a mixed-criticality real-time system. The system-on-chip features FPGA-fabric for bespoke hardware interfaces such as SpaceWire, timers for synchronous camera and illumination triggering and high-speed ADC. A quad-core ARM Cortex A53 processor running Xilinx PetaLinux interfaces to COTS sensors over Gigabit Ethernet, preforms pre-processing such as stereoscopic retrification with OpenCV, and interfaces to the OBC/EGSE over Gigabit Ethernet using the ZeroMQ communication middleware library for request/response and publish/subscribe patterns over TCP/IP.

There are two ARM Cortex R5 real-time processors on the MPSoC. The first core interfaces to embedded devices with real-time requirements such as the ADC polling for the force/torque and tactile sensors, and the pre-processing of raw IMU data coming over a serial line. The other core acts as a gateway to the sensors on the SpaceWire network such as the radar and star tracker. The cores communicate with the Cortex A53 software through shared memory buffers using OpenAMP and the Linux remote processor frameworks.

GNAT Pro Developer is used for real-time software development on the Cortex R5's. For the I3DS project Ada 2012 and SPARK 2014 is used for creating sensor interfaces and control software that is compiled as a static library by GNAT. This library is linked with the main program and programmed to the chip with the Xilinx SDK, and the Xilinx-provided BSP with FreeRTOS is used for multitasking, due to lacking platform support for the GNAT on Xilinx UltraScale+ MPSoC as of 2018. There is an ongoing dialogue with AdaCore to replace FreeRTOS with a bare-board Ravenscar run-time environment.

For reasons of GNAT support for the ARM64/Linux target, and due to the extensive use of OpenCV and other C/C++ libraries, all the software for PetaLinux is written in C/C++ using the Xilinx SDK and the PetaLinux/BitBake tools. GNAT support for ARM64 is expected in 2018, and will allow to replace parts of the C++ code on PetaLinux with Ada/SPARK in the coming SRC follow-up projects.

The proposed industrial presentation will describe the project goals; the challenges of the very high data streams and computational demands; the hardware and architectural design choices, and why the Xilinx Zynq UltraScale+ is a compelling platform for high-performance robotic applications; the system interface design; experiences of using Ada and GNAT Pro together with C/C++ and the Xilinx tools on this new and rather complex platform; and finally, the possible route to space for the system.

Multi-concern Dependability-centered Assurance for Space Systems via ConcertoFLA

Barbara Gallina and Zulqarnain Haider, Mälardalen University Sweden
Anna Carlsson, OHB Sweden
Silvia Mazzini and Stefano Puri, Intecs Italy

Space systems need to be engineered in compliance with ECSS standards and need to ensure a certain degree of dependability. European Cooperation for Space Standardization (ECSS) provides standards for engineering, management and qualification of space systems. More specficaly, ECSS has dedicated standards for product assurance for dependability and safety, i.e. ECSS-Q-ST-30C [2] and ECSS-Q-ST-40C [4] respectively. Assurance of all these properties, require the modelling of various system characteristics and their co-analysis, in order to enable the management of trade-offs between them.

CHESS [1] an open source supporting toolset, which is the result of several R&D projects, promotes a component-based development methodology in which particular emphasis is given to the ability to specify the non-functional properties of components, including critical properties such as real-time and dependability related characteristics. In particular, ConcertoFLA [2] [5], which is part of CHESS toolset, allows users (system architects and dependability engineers) to decorate component-based architectural models with dependability-related information, execute Failure Logic Analysis (FLA) techniques, and get the results back-propagated onto the original model. ConcertoFLA is built on top of Failure Propagation Transform Logic (FPTC) [7] and CHESSFLA [10]. ConcertoFLA is a compositional technique to qualitatively assess the dependability of component-based systems, and partially combines and automatizes traditional safety analysis techniques (e.g., Fault Tree Analysis). Its analysis results have also been exploited to semi-automatically generate arguments fragments for safety assurance [8].

In this paper, we present the customization of the CHESS methodology [11] and ConcertoFLA in the context of the ECSS standards to enable architects and dependability experts to define a system and perform dependability-centered co-analysis for assuring the required non-functional properties of the system according to ECSSS requirements. Figure 1 shows a high-level view of the overall workflow of the proposed customization. The initial step is to define the system by modelling its components and the interactions. Then, the system design is annotated with information specifically related to system reliability, safety and security, associated directly with the involved architectural elements, at the chosen level of architectural detail, and the overall dependability analysis is performed and its results are back propagated. In the next step, the results of the analysis are interpreted for multi-concern e.g., reliability, safety and security. Based on this interpretation a decision is made for introducing the dependability means by refactoring the system. This process is iterated, until the sufficient level of these concerns is met. Further, the paper reports on the usage of the proposed customization in the context of the Attitude Control Systems Engineering (ACS). The ACS plays an important role within satellites, by contributing to maintaining the orientation of the satellite in three-dimensional space and needs to be engineered in compliance with the ECSS standards to ensure a certain degree of dependability. Figure 2 shows a component-based architecture of ACS, where the Sun sensor measurements and angular velocity are provided as an input and the system computes the control torque to be applied on the satellite body to achieve the target attitude.

This work presented is partially supported by the AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems) project [6], whose main objectives were elaborated in [7].

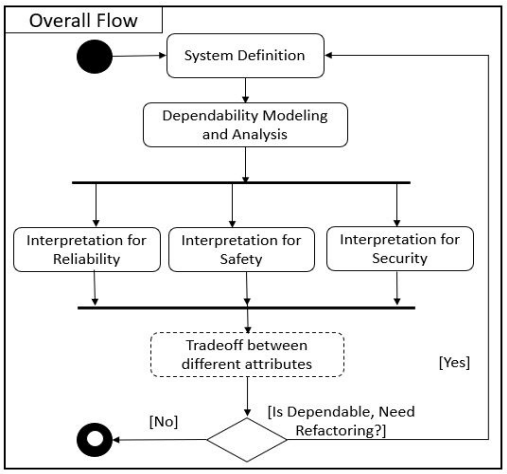


Figure 1 Dependability Co-Analysis via CHESS

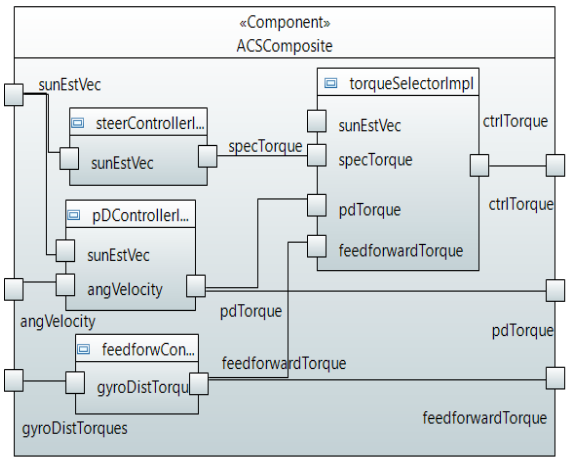


Figure 2 CHESS based architecture of ACS

Acknowledgements

This work is supported by the EU and VINNOVA via the ECSEL project AMASS (No 692474) [6].

References

[1] PolarSys CHESS, https://www.polarsys.org/chess/

[2] Gallina B., Sefer E., Refsdal A. Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis. IEEE International Symposium on Software Reliability Engineering Workshops, Naples, pp. 287-292. 2014.

[3] ECSS-Q-ST-30C Space product assurance – Dependability, March 2009.

[4] ECSS-Q-ST-40C Space product assurance – Safety, March 2009.

[5] CONCERTO Deliverable D3.3 Design and implementation of analysis methods for nonfunctional properties – Final version, 2015.

[6] AMASS, http://www.amass- ecsel.eu.

[7] Ruiz A., Gallina B., de la Vara J.L., Mazzini S., Espinoza H. Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems. Computer Safety, Reliability, and Security. SAFECOMP. LNCS, vol 9923. Springer. 2016.

[8] Alajrami S., Gallina B., Sljivo I., Romanovsky A., Isberg P. Towards Cloud-Based Enactment of Safety-Related Processes. Computer Safety, Reliability, and Security. SAFECOMP. LNCS, vol 9922. Springer. 2016.

[9] Wallace M. Modular architectural representation and analysis of fault propagation and transformation. Electronic Notes in Theoretical Computer Science, volume 141 n.3, pp. 53-71, December, 2005.

[10] Gallina B., Javed M.A., Muram F.U., Punnekkat S. A Model-Driven Dependability Analysis Method for Component-Based Architectures. 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) Cesme, Izmir, pp. 233-240. 2012.

[11] Mazzini S., Favaro J., Puri S., Baracchi L. CHESS: an open source methodology and toolset for the development of critical systems. Third Workshop on Open Source Software for Model Driven Engineering. OSS4MDE. 2016.

Abstract for an industrial presentation:

Author: Andreas Wortmann, OHB System AG, Bremen, Germany

Title: Applying Formal Timing Analysis to Satellite Software

Many functions realized by software in satellite systems are subject to real-time constraints. Such functionality ranges from complex control algorithms including Attitude and Orbit Control to governance of communication bus systems and direct interface and I/O accesses. Even in contingency situations and in the case of hardware anomalies the software execution must be correct and obey all timing constraints in order to ensure safety and reliability of the satellite and its mission.

With increasingly large and complex flight software the traditional approach for ensuring timing-wise correctness of the overall system according to the "look whether it works for all my test cases" approach is not sufficient anymore. This is acknowledged and addressed by the ECSS, calling for a computational model of the software that may serve as a mathematical model for the software schedulability analysis.

Even though the foundation of formal schedulability proofs reaches back to the 1970's, applying the theoretical achievements to commercial satellite software is not yet standard. In the latest missions developed at OHB we have gradually improved awareness and skills in this regard. Being able to efficiently perform timing analysis requires integration of several steps throughout the design and development process, including strict coding, modeling and design rules. Such additional steps need to be in accordance with many other quality and process requirements imposed on flight software.

Roughly speaking, Software Schedulability Analysis consists of three components. A) The tasking model that represents the software comprising tasks, task interactions, the scheduling policy and external triggers. B) Timing information of a large number of code snippets and procedures that are executed by the different tasks. C) A set of constraints that are required to be met. Each of these components impose requirements on the software specification, design and development that are crucial for an analysis to be carried out successfully. The process that hopefully proves the flight software to be compliant with the timing constraints under all conditions engages a set of commercially available and inhouse software tools.

The presentation outlines the methodology to carry out a software schedulability analysis and provides an overview on the techniques and tools successfully engaged at OHB. It presents some lessons learned and summarizes coding and design guidelines. Furthermore, an outlook on possible future achievements with respect to toolchain optimization (e.g. improved developer guidance and "push-button" analysis) and future processor hardware (e.g. multi-core and distributed systems) will be given.

Multicore timing analysis for safety-critical software

Dr. Ian Broster[1], Dr. G. Bernat[1], Dr. F. J. Cazorla[2], C. Evripidou[1], S. Milutinovic[2]

[1] Rapita Systems Ltd. (UK); [2] Barcelona Supercomputing Center (Spain)

In this presentation, we describe challenges and solutions for multicore software timing analysis in a safety-critical aerospace context. We outline a new testing-based approach, gaining traction in industry, for obtaining accurate worst-case execution time measurements from embedded multicore software systems, which can be used in connection with DO-178C and CAST-32A.

Multicore systems are being adopted with greater frequency for safety-critical embedded applications to increase performance and meet the increasing demands of modern embedded software automation. In a safety context such as DO-178C/ED-12C, one of the challenges is to understand and provide assurance for the level of interference between cores and the impact that this has on software performance, particularly that software components complete within their timing budgets despite interference from other (potentially lower criticality/design assurance level) software running on different cores.

We outline an approach for obtaining WCET measurements of critical software, which comprises a detailed analysis of the multicore systems to develop microbenchmarks (opponents) that can be used during testing to analyse the impact of other software on WCET. This involves three main phases:

- In the first phase, both hardware and software components of the multicore platform are investigated, and a semi-automatic process is used to produce specific microbenchmark software programs that adequately test and characterize the impact of multi-core hardware, capturing inter-core resource contention and interference. The microbenchmarks used are based on the qualifiable MµBT (multicore microbenchmark technology) suite by the Barcelona Supercomputing Center and include enhancements for this application.
- The second phase uses Rapita Systems' timing analysis tool Rapi**Time** to collect and analyse execution time data of software obtained while running a suitable selection of microbenchmarks simultaneously. Results from timing analysis and microbenchmark impact analysis are used to either create bounds on the impact of interference or demonstrate that the impact is bounded.
- Finally, safety arguments and evidence are created using a mix of qualified tools and manual processes to feed into the certification documentation and process.

This approach is being used on two commercial aerospace projects during 2018, progress on which (subject to commercial/confidentiality considerations) will be presented.

The method is applicable to Ada and C programs, and the ability to use this method with existing Ada code ported to newer multi-core platforms is an important consideration, although the first industrial applications will be C-based due to the availability of compiler/development environments for the target platforms.

The approach will be further developed through two funded research projects (UK and Catalan authorities), where it will be evaluated in a research context on a further aerospace platform case study. This future work includes a closer integration of the technologies, adding the microbenchmark approach to the Rapi**Time** DO-178C qualification pack and developing the processes for this to be applied smoothly in an aerospace project.

We are also looking to expand its application and gain experience on this approach on further platforms in the avionics domain.

KhronoSim: simulation and testing of real-time critical Cyber-Physical systems

Gonçalo Gouveia [1], João Esteves [1], Cláudio Maia [2], Luis Miguel Pinho [2]
[1] Critical Software, Portugal
[2] CISTER Research Centre, Portugal

Complex systems and systems of systems are an integrated set of components and sub-systems, tightly interacting together to achieve a specific goal. While guaranteeing that individual sub-systems behave according to their specifications is a (relatively) "simple" task, the magnitude of the validation largely increases when it comes to providing guarantees on the correct integrated behaviour.

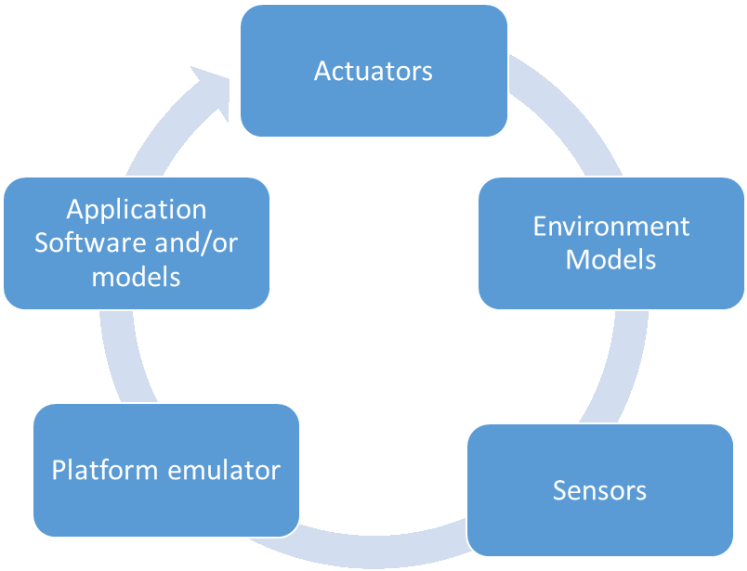
As a matter of fact, all the possible interactions between the sub-systems must be properly tested in order to capture all the system properties and check compliance with the requirements. In addition, testing a system in its actual environment of operation is sometimes overly expensive and/or too slow, in particular when considering Cyber-Physical Systems (CPS), which are known by their interaction with specific physical environments.

The use of model and platform simulators is spreading around and growing in importance to address the aforementioned issues. Simulators allow for an increase in the productivity of software development, enabling three key features:

1. simultaneous development of software and hardware;
2. testing software before actual hardware exists; and
3. providing an environment for software testing, without requiring actual hardware-in-the-loop.

In this context, KhronoSim[1] is a modular and extensible system for simulation and test of complex systems that enables the integration of simulation models and platform emulators in a closed-loop test environment consisting of physical and virtual systems.

[1] Project KhronoSim, funded by the Portuguese National Innovation Agency (ANI) under the ERDF (European Regional Development Fund) through "Portugal 2020" program (017611, POCI-01-0247-FEDER-017611)



The main objectives of KhronoSim are:

- to make it possible to simulate complex systems in real-time by including either the whole or part of the system under test, including the simulation of the environment and other interacting systems;
- to provide a distributed testing architecture, to enable real-time stimuli of the inputs of the system under testing;
- to make it possible to emulate the hardware platform upon which the system will execute, specifically by assuming that it is a multicore embedded platform.

The presentation will present the challenges and approach of KhronoSim, as well as the main building blocks of the framework. It will also discuss how emulation of multicore platforms is integrated within the overall system.

Title

/ Guidelines Compliance and Deviations (the MISRA and CERT cases)

Author

Maurizio Martignano

Affiliation

Spazio IT – Soluzioni Informatiche, Via Manzoni 40, 46030 San Giorgio di Mantova, Italy; Tel: +39 0376

Extended Abstract

Guidelines such as MISRA C:2012 (“guidelines for the use of the C language in critical systems”) and SEI CERT C (“SEI CERT C Coding Standard”) define a set of rules and directives – MISRA (or rules and recommendations - CERT) designed to help developers in writing quality code, i.e. code that is safer, more secure, understandable and maintainable.

Obviously, it is not always possible to adhere to all these guidelines and this why in several software development projects deviations and compliance levels are established in the context of so called customization or tailoring activities.

The customization activity can be performed very formally, adhering once again to specific and additional standards, or rather informally on a project by project base, according to the specific needs and actual limitations of the project itself. The customization strategies adopted by actual projects range from not allowing any deviation to accepting every documented deviation.

Both MISRA and CERT have tried to improve this situation. MISRA in April 2016 has published a relatively new guidance, MISRA Compliance:2016 (“Achieving compliance with MISRA Coding Guidelines). CERT C Coding Standard (as well as the C++ and Java ones) in the Introduction, section 1.8 Conformance, provides a set of indications about the rules and recommendations conformance, defines a set of levels of conformance and describes a proper deviation procedure.

The presentation will start by describing some examples of customizations, implemented by actual projects, and in particular will try to show that:

1. excessive tailoring of the guidelines could be a problem;
2. it is not always true that the knowledge of the language available inside projects is better than the one contained, implied by the guidelines;
3. guidelines can be used to improve the understanding of the language among unexperienced developers.

The provided examples concentrate on usually critical areas like data handling, data representation, defensive programming and (pointer) type conversions.

Secondly, the presentation will describe in detail both MISRA and CERT standardization efforts in the areas of compliance and deviations. Special emphasis will be put on the guidelines prioritization schemes proposed by the two organizations.

Thirdly the presentation will concentrate on the analysis tools and on their importance in verifying, endorsing and encouraging compliance to the guidelines.

Finally, the presentation will propose a viable approach to properly manage compliance and deviations by respecting the new MISRA and CERT indications while retaining the necessary flexibility at project level. This approach is based on concentrating more on actual feasibility rather than following a strict, a priori, adherence to regulations and standards.

Proper emphasis will be put on the educational value of MISRA and CERT guidelines.

Title: Agile in safety critical projects

Authors: Paweł Zakrzewski and Janusz Narkiewicz

From: GE Aviation, Engineering Design Center, Warsaw and Warsaw University of Technology

Abstract:

Almost all avionics software is qualified as safety critical so the development process is a strictly regulated subject including certification which is usually pursued according to the DO-178C standard.

The standard does not define the process of the software development but objectives to be met. DO-178C requires the creation of detailed documentation and assurance of full requirements traceability. Current Agile methodologies expressed by variety of frameworks are being widely used for not safety critical software development and seems to be more effective than traditional waterfall in addressing possible requirement modifications. However, Agile frameworks do not address the notion of requirements traceability, focusing mostly on the delivery of functional features. As such, Agile methodology does not identify documentation creation as an essential part of the project while it is a necessary part of the avionics product certification. For this reason, popular frameworks seem to have limited application for the safety critical aviation products.

In this context, creation and implementation of Agile framework that addresses all safety critical software limitations and fully supports the DO-178C standard would be a breaking point for the industry. The software engineering team at Engineering Design Center works to define the Agile framework that would be successfully implemented for the safety critical software development. It would allow to reduce time and cost of new products development which is a key constrain for the introduction of new avionics technologies.

Review of Agile application potential constraints may allow to compile a list of challenges to be solved by the framework and recognize major obstacles, whether they are related to formal requirements, customers, our organization and culture or internal fear of failure.

An analysis of various software levels occurrences on the avionics market may help to understand what fraction of projects could be addressed considering appropriate DO-178C objectives. According to references more than 60% of the software in Avionics is level C or D. This means that building an efficient Agile framework that works up to level C would address most of the market needs. As the requirement of satisfying DO-178C objectives with independence starts from level B such framework allows to apply development teams organization exactly as it is defined by Scrum. Development team members are able to review the feature internally in the team without necessary engagement of external resources.

One potential obstacle that needs to be taken into account is the planning process of DO-178C. The challenge is how to keep the Software Development Plan (SDP), and Plan for Software Aspects of Certification (PSAC) up to date with incremental and iterative development of the product and its

architecture. Strategy to keep these processes out of Agile framework may work but it is questionable if we can build proper definition of done in this case. Smart integration of the planning process with Agile software development may be feasible with support from dedicated teamwork tools.

This industrial presentation is an attempt to share our experiences and ideas for future deployment.

AGILE-R: Agile Software Development for Railways

J. Favaro, G. Ioele, G. Gennaro, S. Mazzini, P. Panaroni, U. Paone,

Intecs Solutions, Italy

Agile approaches have their roots in 2001 with the elaboration of the famous “Agile Manifesto”. Subsequently, it gradually gained in popularity over the so called “heavy processes” (waterfall based, the champion being CMMI) and Agile is now the most adopted software development approach worldwide. Even the proponents of CMMI are now proposing a “marriage” with Agile [3].

However, for over a decade there has been much controversy with respect to Agile in real time and safety critical software domains such as avionics, space, railway, automotive, medical devices, etc. Today, however, there are many success stories and considerable experience proving that Agile is not in contradiction with highly critical software, on the condition that agility is applied with rigor and discipline.

“Barriers to using Agile no longer exist. Developments in globally distributed teams, large projects, safety-critical systems, and hardware and systems engineering have shown that Agile technologies are adoptable and adaptable.” is reported in *IEEE Software Magazine* [10]. An Agile Software Development Handbook [5] has been developed by the European Cooperation for Space Standards (ECSS), with the support of the European Space Agency, providing detailed guidelines and advice for the adoption of the Scrum software development approach in those space projects where ECSS-E-ST-40 and ECSS-Q-ST-80 are applicable. Detailed mappings between requirements and agile practices have been reported for the avionics sector, while adoption of agile development methods is reported by the NASA Ames Research Center for the development of mission control technology software [6]. Recently a Norwegian study from SINTEF ICT and NTNU has proposed Agile for CENELEC EN 50128, by defining the SafeScrum variant of Scrum with some the CENELEC requirements outside of the agile approach and some safety requirements added to the agile methodology, together with the involvement of the assessor as early as possible to reduce certification costs [8].

In this paper we present AGILE-R, a practical approach adopted by Intecs Solutions for the application of Scrum to the development of Railway software in accordance with the EN 50128 standard (at least up to criticality SIL2) [1]. The main goals of AGILE-R are:

- to reduce time to market and improve responsiveness to change, without sacrificing safety and quality. This is mainly achieved by shortening the time between development and bug fixing. Every increment is fully tested. Regression risks are also reduced. While Agile does not reduce the number of total tests to be executed, it distributes them over manageable quantities for any given increment. In this way, the high costs and delays associated with large and late integration of software are avoided.
- to increase the control and predictability of the development process itself, by forcing visible and tangible results at fixed intervals. Progress is measured by the state of the product rather than estimations and presentations.
- to decrease the risk of producing unsatisfactory solutions, with strong involvement of the product owner.

AGILE-R has been elaborated by an Intecs Solutions team combining diverse and complementary sets of expertise, including Software Methodologies, Safety Assessments, Quality Assurance, CENELEC Standards, Agile, Scrum, and Project Management. The results have been shared and discussed with external Independent Safety Assessors.

AGILE-R in practice

A case study for the application of AGILE-R was defined in the context of the Railway Sirio-LX prodct. Sirio-LX is an automatic radar-based system for preventing trains from colliding with obstacles on the track at level crossings. Sirio-LX is designed to ensure the highest level of safety standard CENELEC SIL4.

The experiment has been the development of a software part outside the official development, not starting from scratch, with the execution of 1 week initial planning phase and Sprint 0 and 2 Sprints with time box of 3 weeks. The main goal was to tune the approach getting learning lessons «from the battlefield» and remove some skepticism.

The impact on EN50128 planning phase was minimal, the approach was welcome by the team with no resistance. Education on Agile and Scrum principles was straightforward. Globally the staff reported a positive experience. Respect of budget and schedule was maintained.

The use of AGILE-R has confirmed the following:

- Agile is a way to manage the software development life cycle, not a different standard.
- Agile does not impose specific new work products, and all documents of EN 50128 have been adopted to ensure compliance.
- There are no contradictions between the application of AGILE-R and formal assessments.
- Agile does not sacrifice safety and quality (these are even better thanks to early detection of bugs and pair programming).
- Only few adaptations have been recommended to best apply AGILE-R with the right balance of agility and discipline.
- Project pitfalls, such as wrong or simplistic design, poor tools, and immature test environment, have an impact in the same way as with the traditional approach but you learn it after a short period of time and you can implement some counter-measures in the early stages of the project.

References

[1] CENELEC EN 50128:2011Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems, 2011.

[2] Balancing Agility with Discipline, A Guide for the perplexed, Barry Boehm, Richard Turner, Addison Wesley, ISBN 0-321-18612-5

[3] CMMI & Agile: why not embrace both? CMMI Institute.

[4] CENELEC 50128 and IEC 62279 Standards, Jean Louis Boulanger (chapter 7.5), book

[5] ECSS-EHB-40A Software engineering handbook, 2013

[6] J. Trimble, C. Webster, Agile Development Methods for Space Operations, Proceedings of SpaceOps 2012 Conference, 2012.

[7] S. H. VanderLeest, A. Buter, Escape the Waterfall: Agile for Aerospace, Proc of the IEEE/AIAA 28th Digital Avionics Systems Conference, 2009.

[8] T. Myklebust, T. Stålhane, N. Lyngby, Application of an Agile Development Process for EN50128/railway conformant Software, Proceedings of the 25th European Safety and Reliability Conference, 2015.

[9] C. Scholz, Agile Software Development compliant to Safety Standards?, Proceedings of the 19th Ada Europe International Conference on Reliable Software Technologies , 2014.

[10]C. Ebert, M. Paasivaara, Scaling Agile - IEEE Software 2017 (issue 6), p. 98

ORGANIZATION

General Chair

Nuno Neves
LASIGE/U. Lisboa, Portugal

Program Chair

António Casimiro
LASIGE/U. Lisboa, Portugal

Special Session Chair

Marcus Völp
University of Luxembourg, Luxembourg

Tutorial and Workshop Chair

David Pereira
CISTER/ISEP, Portugal

Industrial Co-Chairs

Marco Panunzio
Thales Alenia Space., France

José Rufino
LASIGE/U. Lisboa, Portugal

Publication Chair

Pedro Ferreira
LASIGE/U. Lisboa, Portugal

Exhibition Co-Chairs

José Neves
GMV Skysoft, Portugal

Ahlan Marriott
White Elephant GmbH, Switzerland

Publicity Chair

Dirk Craeynest
Ada-Belgium & KU Leuven, Belgium

Local Secretariat

Madalena Almeida
Viagens Abreu, Portugal

Program Committee

Mario Aldea (Universidad de Cantabria), Ezio Bartocci (Vienna University of Technology), Johann Blieberger (Vienna University of Technology), Rakesh Bobba (Oregon State University), Bernd Burgstaller (Yonsei University), António Casimiro (LASIGE/U. Lisboa), Juan A. de la Puente (Universidad Politécnica de Madrid), Virgil Gligor (Carnegie Mellon University), Michael González Harbour (Universidad de Cantabria), J. Javier Gutiérrez (Universidad de Cantabria), Jérôme Hugues (ISAE), Ruediger Kapitza (Technische Universität Braunschweig), Hubert Keller (Karlsruhe Institute of Technology), Raimund Kirner (Univ. of Hertfordshire), Adam Lackorzynski (TU Dresden and Kernkonzept GmbH), Kristina Lundkvist (Mälardalen University), Franco Mazzanti (ISTI-CNR), Laurent Pautet (Telecom ParisTech), Luís Miguel Pinho (CISTER/ISEP), Erhard Plödereder (Universität Stuttgart), Jorge Real (Universitat Politècnica de València), José Ruiz (AdaCore), Sergio Sáez (Universitat Politècnica de València), Elad Schiller (Chalmers University of Technology), Frank Singhoff (Université de Bretagne Occidentale), Jorge Sousa Pinto (University of Minho), Tucker Taft (AdaCore), Elena Troubitsyna (Åbo Akademi University), Santiago Urueña (GMV), Tullio Vardanega (Università di Padova), Marcus Völp (University of Luxembourg).

Industrial Committee

Ian Broster (Rapita Systems), Luís Correia (EMPORDEF-TI), Dirk Craeynest (Ada-Belgium & KU Leuven), Thomas Gruber (Austrian Institute Of Technology - AIT), Andreas Jung (European Space Agency), Ismael Lafoz (Airbus Defence and Space), Ahlan Marriott (White Elephant GmbH), Maurizio Martignano (Spazio IT), Marco Panunzio (Thales Alenia Space), Paul Parkinson (Wind River), Jean-Pierre Rosen (Adalog), José Rufino (LASIGE/U. Lisboa), Emilio Salazar (GMV), Helder Silva (EDISOFT), Jacob Sparre Andersen (JSA Consulting), Andreas Wortmann (OHB System).

CONFERENCE SPONSORS

Springer Verlag publishes the proceedings of the conference, in the Lecture Notes in Computer Science series (LNCS 10873)