MODEL DRIVEN DEVELOPMENT FOR MEDICAL DEVICES IN AADL/BLESS AND SPARK/ADA: PCA PUMP PROTOTYPE

by

Jakub Jedryszek

B.S., Wroclaw University of Technology, Poland, 2012

B.A., Wroclaw University of Economics, Poland, 2012

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2014

Approved by:

Major Professor
John Hatcliff

Copyright

Abstract

Ada programming language is targeted at embedded and real-time systems.

SPARK/Ada is designed for the development of safety and security critical systems. It contains properties, which allows to prove corectness of program and its entities.

AADL (Architecture Analysis & Design Language) is modeling language for representing hardware and software. It is used for real-time, safety critical and embedded systems.

BLESS (Behavior Language for Embedded Systems with Software) is AADL annex sub-language defining behavior of components. The goal of BLESS is automatically-checked correctness proofs of AADL models of embedded electronic systems with software.

Nowadays, we have trend to generate code from models. The ultimate goal of research, which this thesis if part of, is to create AADL/BLESS to SPARK/Ada traslator. Ultimatelly there will be standardized AADL/BLESS models, which will be generating code base for developers extensions (like sceleton code for some Web Framework).

This thesis propose mapping from AADL/BLESS to SPARK/Ada. As an example of Medical Device, PCA Pump (Patient Controlled Analgesia) is used. The foundation for this work is System Requirements for "Integrated Clinical Environment Patient-Controlled Analgesia Infusion Pump System Requirements" (DRAFT 0.10.1) [Lar14] and AADL Models with BLESS annexes created by Brian Larson. Additionally, there was a contribution made in clarifying the requirements document and extending AADL models.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Say thank you for everybody involved directly and indirectly.

Dedication

For my family, mentors and all people who inspired me directly or indirectly in things I am doing.

I also dedicate this thesis to everyone who have supported me throughout the process.

Chapter 1

Introduction

The tale about software safety: why important, software everywhere, human life, etc. (info from 890).

Software Engineering for Real-Time and Safety-Critical systems is very different than creating Desktop applications. In both types of software we want to ensure correctnes and security. In case of e.g. e-mail client software assurance is not crucial. When something happend, we just restart the app. However, in case of e.g. Airplane, software cannot just crash. If it crashes, then people dies. Behind these reasons, we need different properties of our programming language and its tools. For Web or Mobile apps our priority is Rapid Development. For Safety-Critical systems, security is crucial.

Most important in Safety-Critical Systems: Hazard analysis (avoid, recover)! Hazard can cause: * Incident * Accident Accident - event, which cause loss (undesired) Incident - event, which not cause loss (but undesired) Hazard + Environmental Conditions => Accident (loss) Event - state change

1.1 Motivation

There are many accidents where Medical Devices are involved. Very often, the reason is the lack of communication between different Medical Devices. [EXAMPLE ACCIDENT] The solution for such a problem is to create "Integrated Clinical Environment" (ICE). SAnToS Lab at Kansas State University is working on Medical Device Coordination Framework (MDCF), which is prototype implementation of ICE.

Devices working under MDCF will need to satisfy some requirements. To make Developer's life easier, the requirements will be not only in documentation, but also in code. The code will be generated from models. Model Driven Development in this case means we will have some base models for medical devices development and developer will extend and customize them. The same like you do File > 'New Java project' in Eclipse, we want to be able to do the same in e.g. GNAT Programming Studio: File > 'New Medical device project'. Model as specification/requirements.

PCA Pump is as an example of Medical Device, which ultimately will work under Medical Device Coordination Framework (MDCF) developed by SAnToS Lab at Kansas State University. Summarizing, we want to be able to have MDCF, which coordinates Medical Devices. Additionally we want set of AADL/BLESS models, which can be automatically translated to SPARK/Ada. These models will be base for Medical Devices Developers, who can extend and adjust them to implement specific devices. Why AADL? Because it describes hardware and software. It allows to validate that the software will work on some device. Why SPARK? Because it is subset, which is easy to deal with it. In the future, when everything will be done (in case of proving perspective) in SPARK, it will (probably) be extended. Maybe finally, there will be no SPARK, but only Ada. Thus for now, SPARK is temporary subset of Ada for reasoning and corectness proving.

1.2 Contribution

Put all piecies together (SPARK, AADL, BLESS?, ICE, PCA Pump) and analyze current state of target technologies. Review PCA Pump Requirements document Implement PCA Pump based on document, by resolving ambiguities and analyzing different implementation possibilities. Then the implementation is sort of proof that, this document might be base for future Infusion Pumps and/or Medical Devices implementations. Analyzed and extended PCA Pump AADL models, then based on available resource propose possible translation from AADL/BLESS to SPARK/Ada. Created AADL/BLESS to SPARK/Ada translator.

1.3 Organization

The thesis is organized in 6 chapters. Chapter 1 is the of the problem and summary of contribution which was made. Chapter 2 is Background that gives details about Model Driven Development, SPARK/Ada, AADL/BLESS, ICE and available tools for such environment. Chapter 3 describes the implementation of PCA Pump Prototype. Faced issues and design decisions made. Chapter 4 is about code generation from the model. Chapter 5 summarizes all work which has been done in this thesis. Chapter 6 is the future work that can be done on this topic.

Chapter 2

Background

This chapter is brief introduction of all technologies and tools used in this thesis. It is SPARK/Ada programming language and its tools (GNAT Programming Studio, Sireum Bakar, GNATprove), AADL modeling language, BLESS (AADL annex language). There is also overview of the context in which this work has been made: Integrated Clinical Environment standard (ICE) and PCA Pump (ICE compliant device). This is followed by main topic of the thesis: code generation from AADL and analysis of existing AADL translators (Ocarina, RAMSES).

2.1 Integrated Clinical Environment

Medical devices are safety-critical systems. Medical Devices Coordination Framework is an open, experimental ICE-compliant platform to bring together academic researchers, industry vendors, and government regulators. Medical Devices, which are ICE compliant can be connected to MDCF. It enables Medical Devices cooperation. [add some pictures etc.]

2.2 AADL

AADL stands for Architecture Analysis & Design Language. The aim of the AADL is to allow the description of Distributed Real-Time Embedded (DRE) systems by assembling separately developed blocks. Thus it focuses on the definition of clear block interfaces, and separates the implementations from those interfaces. AADL allows for the description of both software and hardware parts of a system [1].

AADL has its roots in DARPA [2] funded research. The first verision (1.0) was approved in 2004 under technical leadership of Peter Feiler [3]. AADL is develop by SAE AADL committee [4]. AADL version 2.0 was published in January 2009. The most recent version (2.1) was published in September 2012 [5].

AADL is a language for Model-Based Engineering [FG13]. It can be represented in textual and graphical form. There are tools (like Osate 2.2.1), which transforms textual representation into graphical. There is also possiblity to represent AADL in XML (using 3rd party tools). An example AADL model called Thermometer is shown in graphical representation in figure 2.2 and in textual representation in listing 2.1.

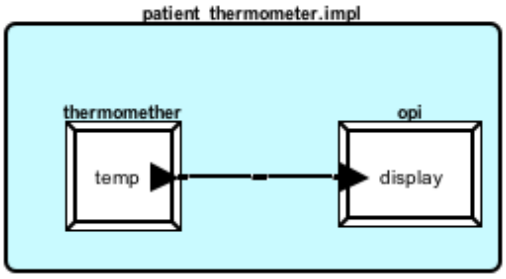

Figure 2.1: *AADL model of simple thermometer*

```
package Thermometer
```

[1] http://penelope.enst.fr/aadl
[2] http://www.darpa.mil
[3] http://wiki.sei.cmu.edu/aadl/index.php/The_Story_of_AADL/
[4] https://wiki.sei.cmu.edu/aadl/index.php/Main_Page
[5] https://wiki.sei.cmu.edu/aadl/index.php/Standardization

```
public
with Base_Types;
  system patient_thermometer
  end patient_thermometer;

  system implementation patient_thermometer.impl
  subcomponents
    thermomether : device thermometer_device.impl;
    opi : device operator_interface.impl;
  connections
    tdn : port thermomether.temp -> opi.display;
  end patient_thermometer.impl;

  device operator_interface
  features
    display : in data port Base_Types::Integer;
  end operator_interface;

  device implementation operator_interface.impl
  end operator_interface.impl;

  device thermometer_device
  features
    temp : out data port Base_Types::Integer;
  end thermometer_device;

  device implementation thermometer_device.impl
  end thermometer_device.impl;
end Thermometer;
```

Listing 2.1: *AADL model of simple thermometer*

Recently AADL becomes a new market standard. There are lots of tools for AADL models analysis, such as: STOOD [6], ADELE [7], Cheddar [8], AADLInspector [9] or Ocarina [10].

What is important, AADL is for architectural description. It should not be compared with UML suites, which allows to link with source code.

2.2.1 OSATE

Open Source AADL Tool Environment (OSATE) is a set of plug-ins on top of the open-source Eclipse platform. It provides a toolset for front-end processing of AADL models. OSATE is developed mainly by SEI (Software Engineering Institute - CMU) [11]. Latest available version of OSATE in the time when this work was published is OSATE2 [12].

2.3 BLESS

BLESS (Behavior Language for Embedded Systems with Software) is AADL annex sub-language defining behavior of components. The goal of BLESS is automatically-checked correctness proofs of AADL models of embedded electronic systems with software.

BLESS contains three AADL annex sublanguages:

- Assertion - it can be attached individually to AADL features (e.g. ports)
- subBLESS - can be attached only to subprograms; it has only value transformations and Assertions without time expressions
- BLESS - it can be attached to AADL thread, device or system components; it contains states, transitions, timeouts, actions, events and Assertions with time expressions...

[6] http://www.ellidiss.com/products/stood
[7] https://wiki.sei.cmu.edu/aadl/index.php/Adele
[8] http://beru.univ-brest.fr/ singhoff/cheddar
[9] http://www.ellidiss.com/products/aadl-inspector
[10] http://www.openaadl.org
[11] http://www.aadl.info/aadl/currentsite/tool/osate.html
[12] https://wiki.sei.cmu.edu/aadl/index.php/Osate_2

How it fits into the picture. Why it was developed. Corectness prove in AADL + behavior [LCH13], from which we can generate SPARK/Ada code.

2.4 SPARK/Ada

First version of Ada programming language, Ada 83 was designed to meet the US Department of Defence Requirements formalized in "Steelman" document [13]. Since that time, Ada evolved. There were Ada 95, Ada 2005 and now we have Ada 2012 (released in December 10, 2012) [14]. Ada is activelly used in many Real-World projects[15], e.g. Aviation (Boeing [16]), Railway Transportation, Commercial Rockets, Satellites and even Banking. One of the main goals of Ada is to ensure software corectness and safety.

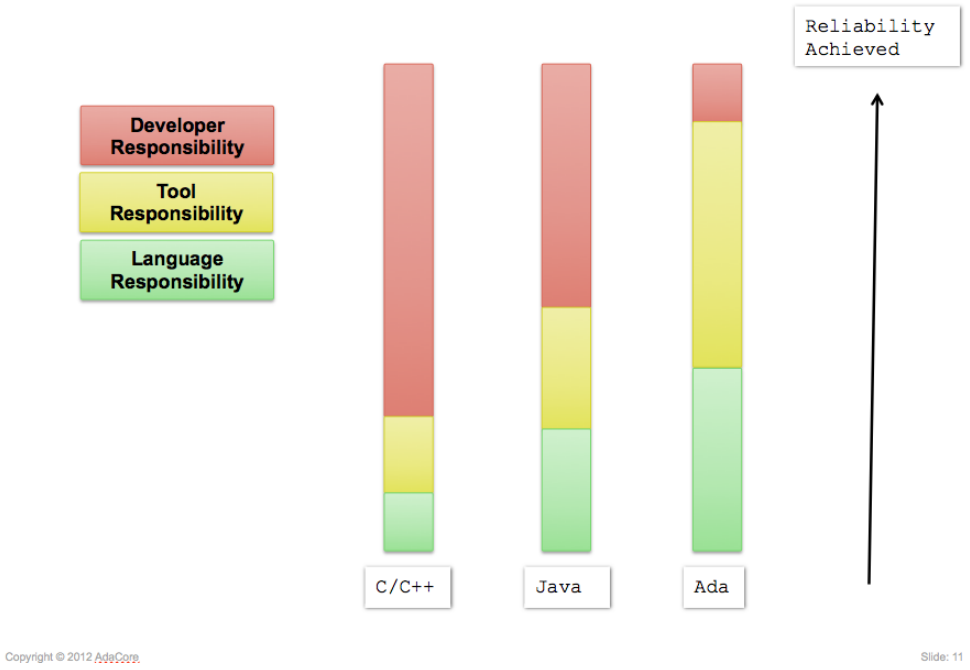

Figure 2.2: *Developer responsibility in Ada*[17].

[13] http://www.adahome.com/History/Steelman/steelman.htm
[14] http://www.ada2012.org
[15] http://www.seas.gwu.edu/ mfeldman/ada-project-summary.html
[16] http://archive.adaic.com/projects/atwork/boeing.html
[17] http://www.slideshare.net/AdaCore/ada-2012

SPARK is a programming language and static verification technology designed specifically for the development of high integrity software. First version was designed over 20 years ago. SPARK has established a track record of use in embedded and critical systems across a diverse range of industrial domains where safety and security are paramount [Bar13].

SPARK provides a significant degree of automation in proving exception freedom [IEC+06]. SPARK excludes some Ada constructs to make static analysis feasible [IEC+06]. Additionally SPARK contains toolset for Software Verification:

- Examiner - analyse code and ensures that it conforms to the SPARK language; also verify program to some extent using Verification Conditions (VC)
- Simplifier - simplify Verification Conditions generated by Examiner
- Proof Checker - prove the Verification Conditions

First version of SPARK was based on Ada 83. SPARK 2005 is based on Ada 2005. It includes annotation language that support flow analysis and formal verification. Annotations are encoded in Ada comments (via the prefix `--#`). It makes every SPARK 2005 programm, valid Ada 2005 program. Figure 2.2 shows example SPARK 2005 package specification.

```
package Odometer
--# own Trip, Total : Integer;
is
  procedure Zero_Trip;
  --# global out Trip;
  --# derives Trip from ;
  --# post Trip = 0;

  function Read_Trip return Integer;
  --# global in Trip;

  function Read_Total return Integer;
  --# global in Total;
```

```
   procedure Inc;
   --# global in out Trip, Total;
   --# derives Trip from Trip & Total from Total;
   --# post Trip = Trip~ + 1 and Total = Total~ + 1;

end Odometer;
```

Listing 2.2: *SPARK 2005 code: Odometer [Bar13]*

SPARK 2005 is a subset of Ada 2005 with annotations. It does not include constructs such as pointers, dynamic memory allocation or recursion [IEC+06].

SPARK 2014 [18] is based on Ada 2012 programming language targeted at safety- and security-critical applications [DEL+14]. Since Ada 2012 contains contracts, there is no need to use annotations like in SPARK 2005. Thus SPARK 2014 is subset of Ada 2012. It contains all features of Ada 2012 except:

- Access types (pointers)
- Exceptions
- Aliasing between variables
- Concurrency features of Ada (Tasking)
- Side effects in expressions and functions

SPARK 2014 does not contains Examiner. Instead, proofs are made by gnatPROVE. The notion of executable contracts in Ada 2012, was inspired by SPARK. Previous Odometer example in SPARK 2014 is shown in figure 2.3.

```
package Odometer
with SPARK_Mode
Abstract_State => (Trip, Total)
is
```

[18] http://www.spark-2014.org

```
   procedure Zero_Trip
   with Global => (Output => (Trip)),
      Depends => (Trip => null),
      Post => (Trip = 0);

   function Read_Trip return Integer
   with Global => (Input => (Trip));

   function Read_Total return Integer
   with Global => (Input => (Total));

   procedure Inc
   with Global => (In_Out => (Trip, Total)),
     Depends => (Trip => Trip, Total => Total),
     Post => Trip = Trip'Old + 1 and Total = Total'Old + 1;

end Odometer;
```

Listing 2.3: *SPARK 2014 code: Odometer*

It is possible to mix SPARK 2014 with Ada 2012. However, only the part which is SPARK 2014 compliant will be verified.

The most popular IDE for SPARK/Ada is GNAT Programming Studio [19].

There is also plugin for Eclipse: GNATbench [20] created by AdaCore. Tools for corectness proving.

2.4.1 GNAT Programming Studio

IDE for SPARK/Ada programs development. Includes proving tools. E.g. Sireum Bakar (developed by SAnToS lab) or GNATprove.

[19] http://libre.adacore.com/tools/gps
[20] https://www.adacore.com/gnatpro/toolsuite/gnatbench/

2.4.2 Sireum Bakar

Overview: symbolic execution, Pilar, Kiasan and Alir [Thi11]. Sireum Kiasan [BHR+11] is a tool, which use symbolic execution for finding possible paths in program. Plugin for GNAT Programming Studio. Plugin for Eclipse (but only SPARK 2005).

2.4.3 GNAT Prove

GNATprove [21] is a formal verification tool for SPARK/Ada programs. It interprets SPARK/Ada annotations exactly like they are interpreted at run time during tests.

2.4.4 AUnit(remove it?)

Overview AUnit tutorials [Fal14] AUnit Cookbook [Ada14]

2.5 PCA Pump

Description of PCA Pump, its functions, problems and how ICE can solve them.

2.6 AADL/BLESS to SPARK/Ada code generation

The ultimate goal of long term research, this thesis is part of is AADL (with BLESS) to SPARK/Ada translation.

2.6.1 Ocarina

Ocarina [LZPH09, LZPH09] generates code from an AADL architecture model to an Ada application running on top of PolyORB framework. In this context, PolyORB acts as both the distribution middleware and execution runtime on all targets supported by PolyORB.

[21] http://www.open-do.org/projects/hi-lite/gnatprove/

It generate Ada 2005 and C code. Since mid-2009, Telecom ParisTech is no longer involved in Ocarina, and is developping another AADL toolchain, based on Eclipse, codenamed RAMSES [Hug13].

2.6.2 Ramses

RAMSES is a model transformation framework dedicated to the refinement of AADL models.

Chapter 3

PCA Pump Prototype

Overview of PCA Pump and issues, which MDCF/ICE will solve.

3.1 PCA Pump Requirements Document

Selected use cases for implementation?

3.2 PCA Pump AADL/BLESS Models

Selected modules for implementation. Pictures etc.

3.3 BeagleBoard-XM

First step was create PCA Pump prototype on BeagleBoard-xM.

BeagleBoard-xM is Embedded device with AM37x 1GHz ARM processor (Cortex-A8 compatible). It has 512 MB RAM, 4 USB 2.0 ports, HDMI port, 28 General-purpose input/output (GPIO) ports and Linux Operating System (on microSD card). All these properties makes this device good candidate for prototyping PCA Pump.

Expansion port 14 and 28 GPIO158 Java Program to Run the pump for 10 seconds

There is no existing SPARK/Ada compiler running on ARM system. Hence, to compile SPARK/Ada program for ARM device, we need to perform cross-compilation on other machine. There is GNAT compiler [Hor09] created by AdaCore, but there was no cross-compiler for ARM. However AdaCore was working on it. They had working version in 2013, but tested only on their target, Android-based device. BeagleBoard-xM is coming with Linux Angstrom Operating System. There is possibility to install Android on BeagleBoard-xM, but still not warranty everything will be working. Cooperation with AdaCore allowed to cross-compile SPARK/Ada program for BeagleBoard-xM.

Include source of simple program? GNAT cross-compiler only for Linux Platform (cross-compilation has to be done on Linux).

3.4 PCA Pump Prototype Implementation

Currently SPARK 2014 does not support tasking [AL14]. For SPARK 2005, GNAT compiler provides Ravenscar Profile [Tea10]. It provides a subset of the tasking facilities of Ada95 and Ada 2005 suitable for the construction of high-integrity concurrent programs.

Issues: Ravenscar Profile, how to deal with different boluses (look at UMinn requirements and annotations for our doc). Look at annotated PCA Pump Req document.

Chapter 4

AADL/BLESS to SPARK/Ada translation

First step was to create mock (based on doc, aadl models and implemented PCA Pump). Prototyping Embedded Systems using AADL lasts for a few years [CB09].

4.1 AADL/BLESS to SPARK/Ada mapping

Mapping is driven by "Architecture analysis & Design Language (AADL) V2 Programming Language Annex Document" [SCD13]. Ocarina tool suite (based on older AADL annex documents [HZPK08]) was also helpful in understanding of AADL to Ada translation. Only high level mapping is done. No implementation (thread interactions) like Ocarina does.

4.1.1 Data types mapping

During AADL/BLESS to SPARK/Ada types mapping, SPARK Examiner was helpful. It detected redundancy in enumerators. Both `Alarm_Type` and `Warning_Type` contained `No_Alarm` enumerators, which was a bug. `Warning_Type` should have `No_Warning` enumerator instead.

4.1.2 AADL ports mapping

Proposed ports mapping shown in table 4.1 is based on AADL runtime services from Annex 2 to "Programming Language Annex Document" [SCD13]. Additionaly, the mapping contains SPARK 2005 contracts.

Table 4.1: AADL to SPARK ports mapping.

AADL/BLESS	SPARK/Ada

`Port_Name :`
`  in data port Port_Type;`
	`-- spec (.ads):`
	`procedure Receive_Port_Name;`
	`--# global out Port_Name;`
	
	`-- body (.adb):`
	`Port_Name : Port_Type;`
	
	`procedure Receive_Port_Name`
	`is`
	`begin`
	`  -- TODO: implement receiving Port_Name value`
	`  -- e.g.:`
	`  -- Port_Name := Some_Pkg.Get_Port_Name;`
	`  null;`
	`end Receive_Port_Name;`

Continued on next page

Table 4.1 – continued from previous page

AADL/BLESS	SPARK/Ada

`Port_Name :`
`  out data port Port_Type;`
	`-- spec (.ads)`
	`function Get_Port_Name return Port_Type;`
	`--# global in Port_Name;`
	
	`-- body (.adb):`
	`Port_Name : Port_Type;`
	
	`function Get_Port_Name return Port_Type`
	`is`
	`begin`
	`  return Port_Name;`
	`end Get_Port_Name;`

Continued on next page

Table 4.1 – continued from previous page

AADL/BLESS	SPARK/Ada

`Port_Name :`
`  in event port;`
	`-- spec (.ads)`
	`procedure Put_Port_Name;`
	
	`-- body (.adb):`
	`procedure Put_Port_Name`
	`is`
	`begin`
	`  -- TODO: implement event handler`
	`  null;`
	`end Put_Port_Name;`

Continued on next page

Table 4.1 – continued from previous page

AADL/BLESS	SPARK/Ada

`Port_Name :`
`  out event port;`
	`-- spec (.ads)`
	`procedure Send_Port_Name;`
	
	`-- body (.adb):`
	
	`procedure Send_Port_Name`
	`is`
	`begin`
	`  -- TODO: implement receiving Port_Name value`
	`  -- e.g.:`
	`  -- Some_Pkg.Put_Port_Name;`
	`  null;`
	`end Send_Port_Name;`

Continued on next page

Table 4.1 – continued from previous page

AADL/BLESS	SPARK/Ada
Port_Name : **in event data port** Port_Type;	-- spec (.ads) **procedure** Put_Port_Name(Port_Name_In : Port_Type); --# global out Port_Name; --# derives Port_Name from Port_Name_In; -- body (.adb): Port_Name : Port_Type; **procedure** Put_Port_Name (Port_Name_In : Port_Type) **is** **begin** Port_Name := Port_Name_In; **end** Put_Port_Name;

Continued on next page

Table 4.1 – continued from previous page

AADL/BLESS	SPARK/Ada
Port_Name : **out event data port** Port_Type;	-- spec (.ads) **procedure** Send_Port_Name; --# global in Port_Name; -- body (.adb): Port_Name : Port_Type; **procedure** Send_Port_Name **is** **begin** -- TODO: implement receiving Port_Name value -- e.g.: -- Some_Pkg.Put_Port_Name(Port_Name); **null**; **end** Send_Port_Name;

4.1.3 Thread to subprograms mapping

Thread features are mapped into SPARK/Ada subprograms according to table 4.2

AADL package, which contains threads is split into child packages with convention: AADL_Package_Name -> AADL_Package_Name.Thread_Name.

In SPARK/Ada we have nested packages and child packages. Sample nested packages are shown in listing 4.1. Equivalent child packages are shown in listing 4.2. The name of a

Table 4.2: *AADL threads to SPARK/Ada subprograms(procedures/functions) mapping.*

AADL/BLESS	SPARK/Ada

package Some_Pkg
  thread Some_Thread
  features
  Some_Port : **out data port** Port_Type;
  end Some_Thread;
end Some_Pkg;
	package Some_Pkg.Some_Thread
	is
	  function Get_Some_Port **return**
	  Port_Type;
	end Some_Pkg.Some_Thread;

package Some_Pkg
  thread Some_Thread
  features
  Some_Port : **out data port** Port_Type
  {Compute_Entrypoint_Source_Text
  => "Custom_Pkg.Custom_Subprogram";};
  end Some_Thread;
end Some_Pkg;
	package Custom_Pkg
	is
	  function Custom_Subprogram
	  return Port_Type;
	end Custom_Pkg;

child package consists of the parent unit's name followed by the child package's identifier, separated by a period (dot) '.'. Calling convention is the same for child and nested packages (e.g. `P.N` in listings 4.1 and 4.2. However, there is a difference between nested packages and child packages. In nested package declarations become visible as they are introduced, in textual order. For example, in listing 4.1 spec `N` cannot refer to `M` in any way. In case of child packages, with certain exceptions, all the functionality of the parent is available to a child and parent can access all its child packages. More precisely: all public and private declarations of the parent package are visible to all child packages. Private child package can be accessed only from parent's body.

```
package P is

   D: Integer;


   --  a nested package:
   package N is

      X: Integer;
```

```
   private
      Foo: Integer;
   end N;

   E: Integer;
private
   --  nested package in private section:
   package M is
      Y: Integer;
   private
      Bar: Integer;
   end M;

end P;
```

Listing 4.1: *Nested packages in SPARK/Ada*

```
package P is
   D: Integer;
   E: Integer;
end P;

--  a child package:
package P.N is
    X: Integer;
  private
    Foo: Integer;
end P.N;

--  a child private package:
private package M is
  Y: Integer;
```

```
private
  Bar: Integer;
end M;
```

Listing 4.2: *Child packages in SPARK/Ada*

4.1.4 Subprograms mapping

I added Subprograms to exisitng PCA Pump AADL models etc. How I did it. Code examples.

Table 4.3: *AADL subprograms to SPARK/Ada subprograms(procedures/functions) mapping.*

AADL/BLESS	SPARK/Ada

subprogram sp
features
  e : **in parameter** T;
  s : **out parameter** T;
end sp;
	procedure sp(e : **in** T; s : **out** T) **is**
	begin
	  null;
	end sp;

data Flow_Rate --dose rate
  properties
  BLESS::Typed=>"integer";
  Data_Model::Base_Type => (
classifier(Base_Types::Integer_16));
  Data_Model::Measurement_Unit => "ml
/hr";
end Flow_Rate;
	subtype Flow_Rate **is** Integer **range** 0 ..
	  Integer'Last;

Table 4.4: *BLESS to SPARK contractsmapping.*

AADL/BLESS	SPARK/Ada
`BLESS::Assertion=>"<<VP()>>"`	`--# pre VP;` `--# post VP;`

`<<Pre()>>Action()<<Post()>>`	**procedure** `Action;`
	`--# pre Pre;`
	`--# post Post;`

4.1.5 BLESS mapping

4.2 "DeusEx" translator

AADL/BLESS to SPARK/Ada translator in Scala. Main idea. Maybe at least create base: AADL to AST covertion?

Chapter 5

Summary

What I have done.

Issues:

- not many online resources - no access to industry code - everything (AADL, SPARK2014, BLESS, tools) is under development - hard to create running application - need to rely on some resources, which are not nessesarly up to date

Chapter 6

Future work

What has to be done now.

translation of BLESS state machine

The semantics of BLESS contain notions of time that make translation to SPARK difficult.

Bibliography

[Ada14] AdaCore. Aunit cookbook. URL: http://docs.adacore.com/aunit-docs/aunit.html, Mars 2014.

[AL14] AdaCore and Altran UK Ltd. Spark 2014 reference manual. URL: http://docs.adacore.com/spark2014-docs/html/lrm, 2011-2014.

[Bar13] John Barnes. *SPARK - The Proven Approach to High Integrity Software*. Altran, 2013.

[BHR+11] Jason Belt, John Hatcliff, Robby, Patrice Chalin, David Hardin, and Xianghua Deng. Bakar kiasan: Flexibe contract checking for critical systems using symbolic execution. In *NASA Formal Methods*, pages 58–72. Springer Berlin Heidelberg, 2011.

[CB09] Mohamed Yassin Chkouri and Marius Bozga. Prototypying of distributed embedded systems using aadl. In *ACESMB 2009, Second International Workshop on Model Based Architecting and Construction of Embedded Systems*, pages 65–79. Springer Berlin Heidelberg, 2009.

[DEL+14] Claire Dross, Pavlos Efstathopoulos, David Lesens, David Mentre, and Yannick Moy. Rail, space security: Three case studies for spark 2014. In *ERTS 2014: Embedded Real Time Software and Systems*, 2014.

[Fal14] Ed Falis. Aunit tutorials. URL: http://libre.adacore.com/tools/aunit, Mars 2014.

[FG13] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL*. Addison-Wesley, 2013.

[Hor09] Bartłomiej Horn. Ada'05 compiler for arm based systems. thesis, Technical University of Lodz, Poland, 2009.

[Hug13] Jérôme Hugues. About ocarina. URL: http://www.openaadl.org/ocarina.html, 2013.

[HZPK08] Jérôme Hugues, Bechir Zalila, Laurent Pautet, and Fabrice Kordon. From the prototype to the final embedded system using the ocarina aadl tool suite. *ACM Transactions on Embedded Computing Systems*, 7(4):237–250, Juilliet 2008.

[IEC+06] Andrew Ireland, Bill J. Ellis, Andrew Cook, Roderick Chapman, and Janet Barnes. An integrated approach to high integrity software verification. *Journal of Automated Reasoning*, 36(4):379–410, Avril 2006.

[Lar14] Brian R. Larson. Integrated clinical environment patient-controlled analgesia infusion pump system requirements draft 0.10.1, Février 2014.

[LCH13] Brian R. Larson, Patrice Chalin, and John Hatcliff. Bless: Formal specification and verification of behaviors for embedded systems with software. In *NASA Formal Methods*, pages 276–290. Springer Berlin Heidelberg, 2013.

[LZPH09] Gilles Lasnier, Bechir Zalila, Laurent Pautet, and Jérôme Hugues. Ocarina : An environment for aadl models analysis and automatic code generation for high integrity applications. In *Reliable Software Technologies – Ada-Europe 2009*, pages 237–250. Springer Berlin Heidelberg, 2009.

[SCD13] SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, and Aerospace Avionics Systems Division. Aerospace

standard - architecture analysis & design language (aadl) v2 programming language annex document, 2013.

[Tea10] SPARK Team. The spark ravenscar profile. URL: http://docs.adacore.com/sparkdocs-docs/Examiner_Ravenscar.htm, 2010.

[Thi11] Hariharan Thiagarajan. Dependence analysis for inferring information flow properties in spark ada programs. thesis, Kansas State University, 2011.

Appendix A

PCA Pump Prototype - simple, working example

Content of this appendix.

Appendix B

PCA Pump Prototype - translated from AADL/BLESS