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 is a language is targeted at embedded and real-time systems.

SPARK is subset/superset (remove some funcionalities from Ada, but add contracts - in 2005, in 2014: only remove some stuff from Ada like no exceptions, no goto) of Ada with Code Contracts. It is designed for High-Assurance systems. It 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 standarized AADL/BLESS models, which will be generating code base for developers extensions (like sceleton code for some Web Framework).

Medical Devices are very important part of High-Assurance systems.

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 Ïntegrated Clinical Environment Patient-Controlled Analgesia Infusion Pump(DRAFT 0.10.1) 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).

1.1 Motivation

Why we need this model driven development. 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'. Why AADL? Why SPARK? Because 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)

1.3 Organization

How many chapters etc.

Chapter 2

Background

General overview of all below things?

2.1 High-assurance systems

What software properties and requirements exists to ensure safety. Medical Devices as type of High-assurance systems.

2.2 SPARK/Ada

History of Ada: http://www.adahome.com/History/Steelman/intro.htm Ada fulfill US DOD requirements. Rationale of this language. Good for Software Verification etc. http://www.slideshare.net/A 2012 (slide 11: dev responsibility) GNAT Programming Studio. Tools for corectness proving.

2.2.1 GNAT Programming Studio

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

2.2.2 Sireum Kiasan

Overview: symbolic execution, Pilar, Alir[THBR12]? Sireum Kiasan[DLR06] 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).

2.2.3 GNAT Prove

Overview

2.2.4 AUnit

Overview

2.3 AADL

Rationale of this language. âĂIJAADLâĂİ 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.

2.3.1 Osate

OSATE is a set of plug-ins on top of the open-source Eclipse platform to provide a toolset for front-end processing of AADL models. It is developed mainly by SEI (Software Engineering Institute - CMU).

2.4 BLESS

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

2.5 Integrated Clinical Environment

http://santos.cis.ksu.edu/MDCF/doc/ICE-Motivation.pdf http://santos.cis.ksu.edu/MDCF/doc/MDCF-Tutorial-Overview.pdf

2.6 PCA Pump

http://www.santoslab.org/pub/paper/LarsonEtAl13-PCA-Requirements-SEHC-preprint.pdf

2.7 AADL/BLESS to SPARK/Ada code generation(maybe translation is better word?)

The ultimate goal of long term research, this thesis is part of. AADLto Ada BLESS to SPARK contracts + (eventually) behavior

2.7.1 Ocarina

Overview: AADL->Ada on PolyOrb, support for other languages?

Ocarina generate 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. It generate Ada 2005 and C code.

2.7.2 Ramses

Overview. Not sure if it is useful at any point here?

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. No gnat compiler at the beginning. Then after cooperation with AdaCore, managed to run Ada program on BB (include source as appendix?). Only Ada (not SPARK), but working! Gnat compiler only for Linux Platform (for now).

3.4 PCA Pump Prototype Implementation

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).

4.1 Extention of exisitng PCA Pump AADL models

I added Subprograms etc. How I did it. Code examples.

4.2 AADL/BLESS to SPARK/Ada mapping

Mapping is driven by Ocarina and "Architecture analysis & Design Language (AADL) V2 Programming Language Annex Document". Only high level mapping is done. No implementation (thread interactions). Table, how specific constructs (subset) in AADL/BLESS are translated to SPARK/Ada.

4.3 "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.

Chapter 6

Future work

What has to be done now.

Bibliography

[DLR06] Xianghua Deng, Jooyong Lee, and Robby. Bogor/kiasan: A k-bounded symbolic execution for checking strong heap properties of open systems. In *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*, pages 157–166, 2006.

[THBR12] Hariharan Thiagarajan, John Hatcliff, Jason Belt, and Robby. Bakar alir: Supporting developers in construction of information flow contracts in spark. *Source Code Analysis and Manipulation, IEEE International Workshop on*, 0:132–137, 2012.

Appendix A

Title for This Appendix

```
dm = 4
nm = 4
tm = 2
dimMAX = dm + 1;
part /: part[0] = {{}};
Do[part /: part[n] =
   Union[Flatten[
     Table[Sort[Join[{i}, #]] & /@ part[n - i], {i, 1, n}], 1]
   1, dimMAX}];
L = (Times @@ # &) /@ Map[c, part[dm], {2}];
```

Appendix B

Title for This Appendix