A MODEL-DRIVEN DEVELOPMENT AND VERIFICATION APPROACH FOR MEDICAL DEVICES

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

Abstract

Medical devices are safety-critical systems whose failure may put human life in danger. They are becoming more advanced and thus more complex. This leads to bigger and more complicated code-bases that are hard to maintain and verify. Model-driven development provides high-level and abstract description of the system in the form of models that omit details, which are not relevant during the design phase. This allows for certain types of verification and hazard analysis to be performed on the models. These models can then be translated into code. However, errors that do not exist in the models may be introduced during the implementation phase. Automated translation from verified models to code may prevent to some extent.

This thesis proposes approach for model-driven development and verification of medical devices. Models are created in AADL (Architecture Analysis & Design Language), a language for software and hardware architecture modeling. AADL models are translated to SPARK Ada, contract-based programming language, which is suitable for software verification. Generated code base is further extended by developers to implement internals of specific devices. Created programs can be verified using SPARK tools.

A PCA (Patient Controlled Analgesia) pump medical device is used to illustrate the primary artifacts and process steps. The foundation for this work is "Integrated Clinical Environment Patient-Controlled Analgesia Infusion Pump System Requirements" document and AADL Models created by Brian Larson. In addition to proposed model-driven development approach, a PCA pump prototype was created using the BeagleBoard-xM device as a platform. Some components of PCA pump prototype were verified by SPARK tools and Bakar Kiasan.

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

Software is present in all aspects of our life. From the simple program in alarm clock to iPad, through cars, refrigerators and computers. Furthermore, our lives are getting more and more depended on Software. Usually when we think about Software, we think about Applications for PC or Smart Phone. E.g. Calculator, Word processor or Stock Market application. In this case, rapid development and smooth operation is a key. However, there is also another, very important class of Software: Safety Critical Systems. It comprises software for Airplanes, Medical Devices, Satellites or Rockets.

Software Engineering for Real-Time and Safety-Critical Systems is very different than creating Business applications. In both types of software we want to ensure correctness and security. However, in each of them, to different extent. In case of mentioned Word processor, software assurance is not critical. When it crashes, it can be restarted. In worst case scenario, some part of work might be lost. Airplane software crash may put human life in danger or even cause the death. Thus for Safety-Critical systems, the security and correctness are crucial. Behind these reasons, different Software Design methodology and different properties of programming language and its tools are needed.

The most important part of Safety-Critical Systems Design is hazard analysis. How to avoid unintentional states and how to recover from them. Hazard can cause incident or accident. Former is an event, which not cause a loss (but undesired), and could lead to accident. Latter cause the loss (and it is also undesired). Hazard analysis can be done manually by human or automatically by software tools. AADL, BLESS and SPARK Ada contains variety of them.

1.1 Motivation

[IF FIRST=BETTER] Nowadays, medical devices works rather independently. It leads to many accidents, which could have been avoided by their interoperability. For example some drug (e.g. morphine), which is delivered by Patient-controlled analgesia (PCA) pump after surgery, can cause low oxygen level or even lack of pulse. That can lead to patient's death. PCA pump does not monitor oxygen level, but Oxygen monitoring device does. If these two devices are organized in centralized system, which implements safety interlock mechanism to shutdown the pump when low oxygen level is detected, accident can be avoided.

In order to communicate, devices have to use compatible interfaces and protocols. There is a concept of "Integrated Clinical Environment" (ICE). It is series of standards, which describes medical device integration and interoperability. SAnToS lab created Medical Device Coordination Framework (MDCF), which is prototype implementation of ICE. Standards are captured not only as requirement documents, but also in software and hardware models form. It allows different medical devices, created by different vendors, to be connected and work under supervision of centralized system.

[ELSE] There are many accidents where Medical Devices are involved. Very often, the reason is the lack of communication between them. Drug dosed by PCA Pump may affect patient's level of oxygen and carbon dioxide level. Thus adequate monitoring of patient's levels of oxygen and carbon dioxide is required. Moreover, integrated system, which will take adequate action in case of hazard is needed. The solution for such a problem is "Integrated Clinical Environment" (ICE). SAnToS Lab at Kansas State University, in cooperation with University of Pennsylvania are working on Medical Device Coordination Framework (MDCF) [HKL+12], which is prototype implementation of ICE. It is an open source framework for coordinating multiple medical devices to work together.

Devices working under MDCF have to satisfy some requirements. To make Developer's life easier, the requirements are not only in documentation, but also in form of software and hardware models. Model Driven Development in this case means that there from base models for medical devices development, from which skeleton code can be generated. Developers extend and customize generated code for specific device. In the same fashion like File > 'New Java project' create code skeleton, File > 'New Medical device project' will create code structure which has to be implemented. The ultimate goal is to create set of models for different medical devices, which can be automatically translated to code.

[END IF]

PCA pump prototype created in this thesis is as an example of Medical Device, which ultimately will work under MDCF.

1.2 Technologies

AADL (Architecture Analysis & Design Language) is modeling language for representing hardware and software. It is used for real-time, safety critical and embedded systems. AADL allows for the description of both software and hardware parts of a system. It is used to describe architecture, but AADL allows to add behavioral extensions through annex languages. 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.

Ada is one of the most popular (along with C/C++) programming language targeted at embedded and real-time systems. SPARK Ada is subset of Ada, designed for the development of safety and security critical systems. It contains subset, which allows to reason about and prove correctness of program and its entities. There are also SPARK tools for software verification.

1.3 Goals

The initial goals, which most of them is accomplished are as follows:

- identify PCA Pump and Infusion pumps properties and internals required for implementation
- SPARK Ada cross-compilation for ARM-device (BeagleBoard-xM)
- implement PCA Pump based on Brian Larson's Requirement Document [LHC13]
- develop AADL/BLESS to SPARK Ada mapping
- mock PCA Pump AADL/BLESS models in SPARK Ada (based on created mapping and implementation)
- implement not generated part (based on implementation) [NOT ACCOMPLISHED - REMOVE?]
- create AADL/BLESS to SPARK Ada translator [NOT ACCOMPLISHED - REMOVE?]
- Use SPARK tool set for software verification:
  - SPARK Examiner
  - SPARK Simplifier
  - Proof Obligation Summarizer (POGS)
  - Bakar Kiasan
  - GNATprove

1.4 Contribution

This thesis demonstrates how AADL/BLESS models can be mapped to SPARK Ada. Additionally it presents current possibilities and limitations of SPARK Ada language, Ravenscar profile and SPARK verification tools. The main contributions of this thesis are as follows:

- Review of PCA Pump Requirements document [LHC13]
- Cross-compilation and testing of SPARK Ada 2005 and 2014 programs on BeagleBoard-xM platform
- Implementation of PCA pump based on Requirements document [LHC13] and AADL/BLESS models, which validates them
- Analysis of different PCA pump implementation possibilities
- AADL/BLESS to SPARK Ada translation schemes
- Practical demonstration of SPARK 2005 verification tools: its capabilities and limitations

1.5 Organization

The thesis is organized in 8 [fix this: how to count all chapters?] chapters:

- Chapter 1 is the problem description and summary of contribution which has been made.
- Chapter 2 is Background that gives details about ICE, MDCF, Model-Driven Development, AADL, BLESS, SPARK Ada and available tools for such environment.
- Chapter 3 describes Patient-Controlled Analgesia (PCA) pump.
- Chapter 4 presents mappings from AADL/BLESS to SPARK Ada.

- Chapter 5 describes the implementation of PCA Pump Prototype. Faced issues and design decisions made.
- Chapter 6 describes verification of implemented PCA Pump Prototype.
- Chapter 7 summarizes all work which has been done in this thesis.
- Chapter 8 is the future work that can be done on this topic.

1.6 Terms and Acronyms

- **AADL** - Architecture Analysis & Design Language
- **BLESS** - Behavioral Language for Embedded Systems with Software
- **ICE** - Integrated Clinical Environment
- **MDCF** - Medical Device Coordination Framework
- **PCA** - Patient-Controlled Analgesia (pump)
- **FDA** - Food and Drug Administration
- **GPS** - GNAT Programming Studio
- **GCC** - GNU Compiler Collection
- **GUI** - Graphical user interface
- **VC** - Verification Condition
- **DPC** - Dead Path Conjecture
- **POGS** - Proof Obligation Summarizer
- **VTBI** - Volume to be infused

- **KVO** - Keep Vein Open

Chapter 2

Background

This chapter is a brief introduction of all technologies and tools used in this thesis. There are: AADL modeling language, BLESS (AADL annex language), SPARK Ada programming language and its verification tools. There is also an overview of the context in which this work has been done: Integrated Clinical Environment (ICE) standard 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

Idea of "Integrated Clinical Environment" (ICE) was initiated by Dr. Julian Goldman from Center for Integration of Medicine & Innovative Technology. The main idea is to create environment of medical devices network. It will allow clinician and software system to make decisions based not only on output from one device, but from all of them together. ICE purpose is to solve current issues with medical devices, which usually operate independently. It requires more human attention and control through checking output of every device manually and then making decision. ICE will make it easier, e.g. by introducing alarms, which can not only indicate problem but also interact with other devices and make decision automatically. E.g. when PCA Pump infuse some drug to patient's vein and Pulse Oximeter detects low oxygen level, ICE can coordinate PCA pump shutdown.

Moreover, ICE comprises components that may be implemented by different vendors. Such components are medical devices and applications to supervise them. Figure 2.1 presents high overview of ICE system. Medical devices (PCA Pump, Respiratory Rate Monitor and Pulse Oximeter) are connected to the system. All of them are monitored and controlled. There is communication between devices and ICE, in order to exchange data between them and Electronic Medical Record (EMR) Database. Informations in EMR comprises drug library, patient's medical records, monitoring logs etc. ICE can make decisions (such as PCA Pump shutdown) based on that informations.

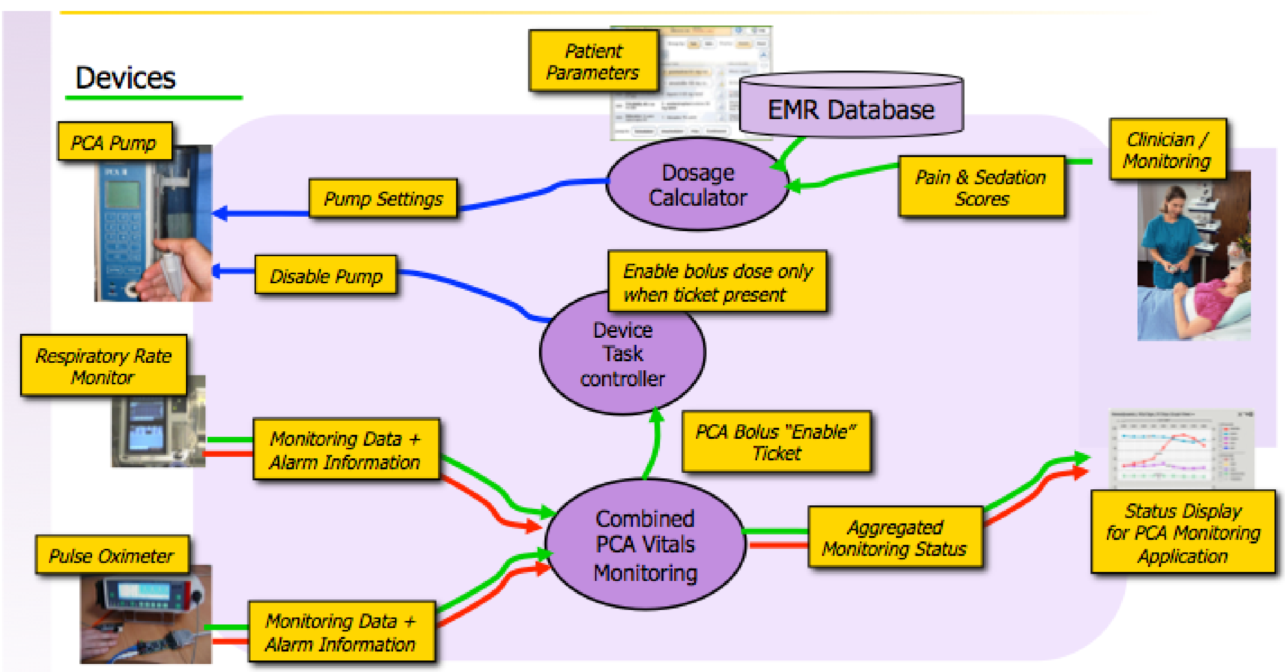

Figure 2.1: *ICE Closed Loop Control*

[ADD MORE INFORMATION?]

2.2 Medical Device Coordination Framework

Medical Device Coordination Framework (MDCF), jointly developed by SAnToS lab (Kansas State University) and University of Pennsylvania is prototype implementation of ICE. It is an open, experimental platform to bring together academic researchers, industry vendors, and government regulators. Project is response to request from Food and Drug Administration (FDA) to build a prototype of ICE. Medical Devices, which are ICE compliant can be connected to MDCF. The framework enables Medical Devices interoperability. MDCF is designed to illustrate by example the issues related to functional concepts, safety, security, verification and certification.

The goals of MDCF project comprises:

- Open source infrastructure
- Meet performance requirements of realistic clinical scenarios
- Provide middleware with reliability, real-time, security
- Provide an effective app programming model and development environment with integrated verification/validation support and construction of regulatory artifacts
- Support evaluation of device interfacing concepts
- Illustrate how to support real and mock devices
- Illustrate envisioned regulatory oversight and 3rd party certification

In this thesis, part of penultimate point will be illustrated. For now, MDCF use only mock devices, which are Java desktop applications. PCA Pump Prototype aim to be first real-device.

MDCF uses publish-subscribe architecture for communication between components: apps and devices. Figure 2.2 presents MDCF structure. Devices, like PCA pump, are clients.

MDCF Server is integration layer which comprises Core and applications working in top of it. [HLW12].

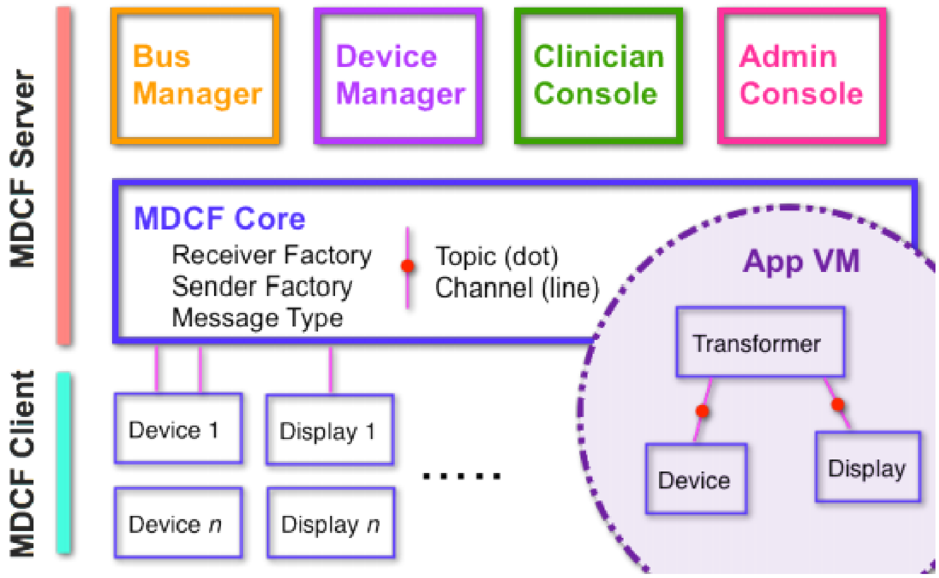

Figure 2.2: *MDCF architecture and example app virtual machine (lower right)*

[ADD MORE INFORMATION?]

2.3 AADL

AADL stands for Architecture Analysis & Design Language. It is used to model embedded and real-time systems. AADL allows for the description of both software and hardware parts of a system. It can be used not only for design phase of software development process, but also for analysis, verification or code generation.

AADL has its roots in DARPA[1] funded research. The first version (1.0) was approved in 2004 under technical leadership of Peter Feiler[2]. AADL is develop by SAE AADL committee[3]. AADL version 2.0 was published in January 2009. The most recent version (2.1)

[1] http://www.darpa.mil
[2] http://wiki.sei.cmu.edu/aadl/index.php/The_Story_of_AADL/
[3] https://wiki.sei.cmu.edu/aadl/index.php/Main_Page

was published in September 2012[4].

AADL is a language for Model-Based Engineering [FG13]. It can be represented in textual and graphical form. There are tools, like OSATE (see section 2.3.1), which transforms textual representation into graphical or XML.

AADL contains entities for modeling software and hardware components. It allows to create interactions and dependencies between them.

AADL Execution Platform Components and Devices:

- Processor / Virtual Processor - Provides thread scheduling and execution services
- Memory - provides storage for data and source code
- Bus / Virtual Bus - provides physical/logical connectivity between execution platform components
- Device - interface to external environment

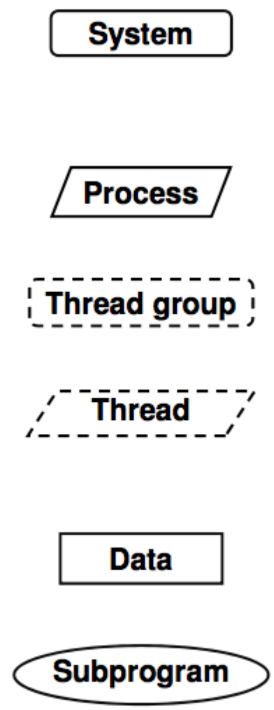

Figure 2.3: *AADL Application Software Components*

Application Software Components of AADL (figure 2.3):

- System - hierarchical organization of components
- Process - protected address space
- Thread group - logical organization of threads
- Thread - a schedulable unit of concurrent execution
- Data - potentially sharable data
- Subprogram - callable unit of sequential code

[4]https://wiki.sei.cmu.edu/aadl/index.php/Standardization

An example AADL model is shown in graphical representation, in the figure 2.4. Its textual representation is presented in listing 2.3.

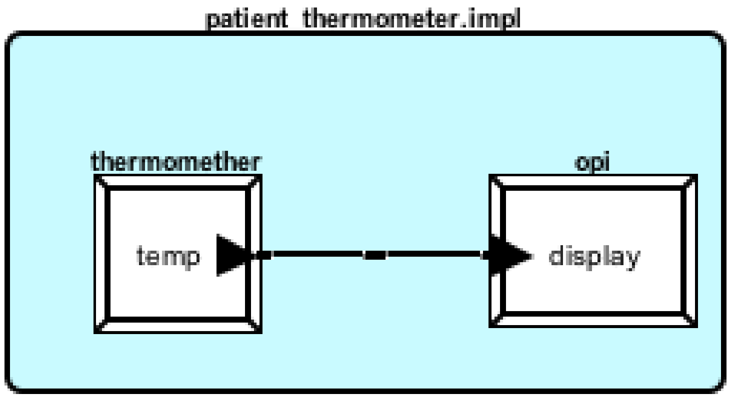

Figure 2.4: *AADL model of simple thermometer*

```
package Thermometer
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: OSATE (see section 2.3.1, STOOD[5], ADELE[6], Cheddar[7], AADLInspector[8] or Ocarina[9].

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

AADL can be extended with the following methods:

- user-defined properties: user can extend the set of applicable properties and add their own to specify their own requirements
- language annexes (the core language is enhanced by annex languages that enrich the architecture description. For now, the following annexes have been defined):
  - Behavior annex: add components behavior with state machines (e.g. BLESS)
  - Error-model annex: specifies fault and propagation concerns
  - ARINC653 annex: defines modelling patterns for modelling avionics system
  - Data-Model annex: describes the modelling of specific data constraint with AADL

More details about AADL can be found in Peter Feiler's book "Model-Based Engineering with AADL" [FG13].

2.3.1 OSATE

Open Source AADL Tool Environment (OSATE) is a set of plug-ins on top of the open-source Eclipse platform. It provides a tool set for front-end processing of AADL models. OSATE

[5] http://www.ellidiss.com/products/stood
[6] https://wiki.sei.cmu.edu/aadl/index.php/Adele
[7] http://beru.univ-brest.fr/ singhoff/cheddar
[8] http://www.ellidiss.com/products/aadl-inspector
[9] http://www.openaadl.org

is developed mainly by SEI (Software Engineering Institute - Carnegie Mellon University)[10]. Latest available version of OSATE in the time when this work was published is OSATE2[11].

OSATE relies on EMF, UML2 and Xtext[12]. It comprises e.g. AADL project wizard, AADL Navigator and AADL syntax analyzer. OSATE enables conversion of AADL in textual representation into graphical. There are also plug-ins for OSATE, like BLESS[13] or OCARINA[14].

2.4 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 sub-languages:

- 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

BLESS annex subclauses can be added to AADL models transparently to other uses of the system architecture. It includes a verification-condition (VC) generation framework and an accompanying proof tool that enables engineers to prove VCs via proof scripts build from system axioms and rules from a user-customizable rule library. [LCH13]

[10]http://www.aadl.info/aadl/currentsite/tool/osate.html
[11]https://wiki.sei.cmu.edu/aadl/index.php/Osate_2
[12]http://www.eclipse.org/Xtext/
[13]http://bless.santoslab.org/node/5
[14]http://libre.adacore.com/tools/ocarina/

The BLESS tool framework is implemented as a publicly available open source plug-in for OSATE (mentioned in section 2.3.1). It includes an editor for BLESS specifications and an environment operating the BLESS proof engine. [LCH13]

Some BLESS constructs can be translated into SPARK contracts, which is part of this thesis. Additionally, BLESS allows to model behavior of components.

[MORE DETAILS? EXAMPLES?]

2.5 SPARK Ada

First version of Ada programming language - Ada 83 - was designed to meet the US Department of Defense Requirements formalized in "Steelman" document[15]. Since that time, Ada evolved. There were Ada 95, Ada 2005 and Ada 2012 (released in December 10, 2012)[16]. Ada is actively used in many Real-World projects[17], e.g. Aviation (Boeing[18]), Railway Transportation, Commercial Rockets, Satellites and even Banking. One of the main goals of Ada is to ensure software correctness and safety. Due to this requirements, Ada minimize developer responsibility in comparison to other programming languages (see figure 2.5). It is achieved not only by language capabilities, but also by tools for verification.

SPARK is a programming language and static verification technology designed specifically for the development of high integrity software. It is a "safe" subset of Ada designed to be susceptible to formal methods, accompanied with a set of approaches and tools. SPARK 2005 does not include constructs such as pointers, dynamic memory allocation or recursion [IEC+06]. Using SPARK, a developer takes a Z specification and performs a stepwise refinement from the specification to SPARK code. For each refinement step a tool is used to

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

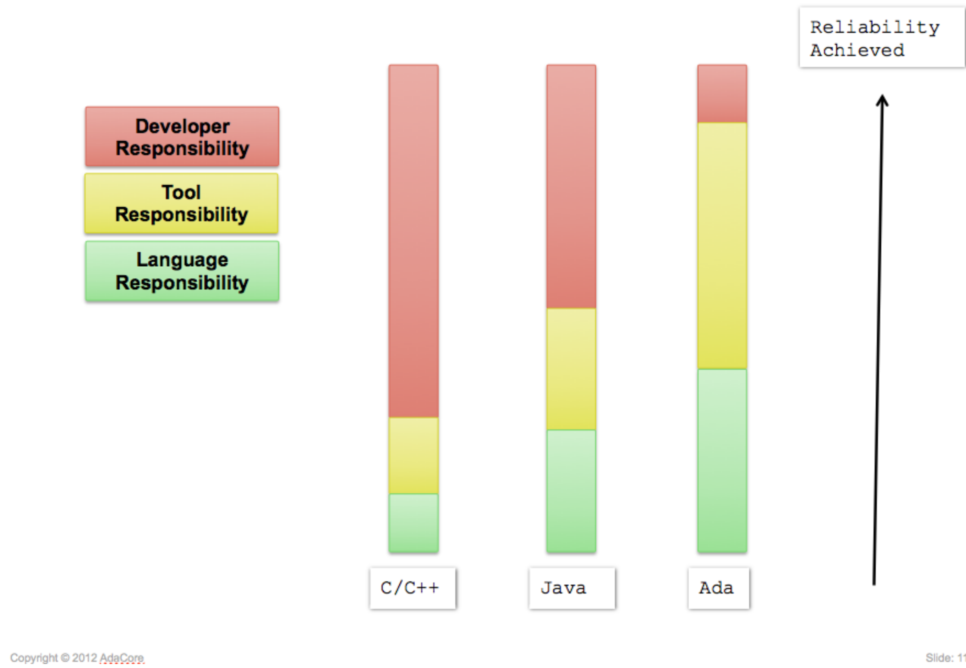

Figure 2.5: *Developer responsibility in Ada*[19].

produce verification conditions (VC's), which are mathematical theorems. If the VC's can be proved then the refinement step will be known to be valid. However if the VC's cannot be proved then the refinement step may be erroneous[20]. Sample Verification Condition contains checks for:

- array index out of range
- type range violation
- division by zero
- numerical overflow

[Add more examples where SPARK is used?]

SPARK provides a significant degree of automation in proving exception freedom [IEC+06]. Some Ada constructs are excluded from SPARK to make static analysis feasible [IEC+06].

[20] http://www.dwheeler.com/lovelace/s17s4.htm

Additionally SPARK contains tool-set for software verification (see section 2.6). In real-world applications, the embedded critical components are written in SPARK while the non-critical components are written in Ada.

First version of SPARK was based on Ada 83. The second version (SPARK 95) - on Ada 95. SPARK 2005 is based on Ada 2005. It is a subset of Ada 2005 with annotations. The annotation language support flow analysis and formal verification. Annotations are encoded in Ada comments (via the prefix `--#`). It makes every SPARK 2005 program, valid Ada 2005 program. SPARK annotations contains code contracts, which are analyzed by verification tools, but ignored by Ada compiler.

Listing 2.5 presents simple procedure with code contracts. It increments variable given as parameter by 1. The `derives` clause specify variable dependency. It future value depends on its current value. There is precondition saying that the value has to be lower than maximum value of `Integer` type. There is also post condition, which states that the value of variable (given as parameter) after the procedure execution has to be equal to its previous value incremented by 1 ('~' attached to variable means value of this variable, before procedure execution).

```
procedure Increment (X : in out Integer);
--# derives X from X;
--# pre X < Integer'Last;
--# post X = X~ + 1;
```

Listing 2.2: *Sample SPARK procedure with code contracts*

SPARK 2014[21] (based on Ada 2012) is under development. There is partial tool support (in GNAT Programming Studio), but some language features are still not supported. It is worth to mention, that Ada 2012 contains code contracts (which was inspired by previous versions of SPARK). Thus SPARK 2014 is just a subset of Ada 2012. [DEL+14] It contains all features of Ada 2012 except:

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

- Access types (pointers)
- Exceptions
- Aliasing between variables
- Concurrency features of Ada (Tasking) - it's part of SPARK 2014 road-map to include support for tasking in the future, although likely not this year
- Side effects in expressions and functions

Table 2.1 presents fundamental SPARK 2005 annotations and their equivalents in SPARK 2014 and Ada 2012.

Table 2.1: Fundamental SPARK annotations

SPARK 2005	**SPARK 2014**	**Description**
`--# global`	`Global`	list of used global variables within subprogram
`--# derives`	`Depends`	describe dependencies between variables
`--# own`	`Abstract_State`	declare variables defined in package body
`--# initializes`	`initializes`	indicates variables, which are initialized
`--# inherit`	not needed	allows to access entities of other packages
`--# pre`	`Pre`	pre condition

Continued on next page

Table 2.1 – continued from previous page

SPARK 2005	SPARK 2014	Description
`--# post`	`Post`	post condition
`--# assert`	`Assert`	assertion

Sample mapping from SPARK 2005 to 2014 is shown in the table 2.2. Complete mapping can be found in SPARK 2014 documentation[22] [AL14a].

Table 2.2: *Sample SPARK 2005 to 2014 mapping.*

SPARK 2005	SPARK 2014
`--# global in out X, Y;`	`with Global  => (In_Out => (X, Y));`
`--# derives X from Y &` `--#         Y from X;`	`Depends => (X => Y,` `             Y => X);`
`--# pre Y /= 0 and` `--#     X > Integer'First;`	`with Pre  => Y /= 0 and` `              X > Integer'First;`
`--# post X = Y~ and Y = X~;`	`with Post => (X = Y'Old and Y = X'Old);`

The previous example (listing 2.5) translated to SPARK 2014 is shown in figure 2.5.

```
procedure Increment (X : in out Integer)
with Depends => (X => X),
  Pre => (X < Integer'Last),
  Post => (X = X'Old + 1);
```

[22] http://docs.adacore.com/spark2014-docs/html/lrm/mapping-spec.html

Listing 2.3: *Sample SPARK 2014 procedure and Code Contracts*

It is possible to mix SPARK 2014 with Ada 2012. However, only the part which is SPARK 2014 compliant will be verified. As mentioned before, usually SPARK is used in the most critical parts of Software Systems [Cha00]. It means, that some part is written in e.g. Ada or C++ and the rest in SPARK. The reason of that is the SPARK limitation and lack of necessity to verify some not safety-critical modules. SPARK 2014 does not contains Examiner like SPARK 2005. Instead, proofs are made by gnatPROVE (see section 6.5).

The most popular IDE for SPARK Ada is GNAT Programming Studio[23] (see section 2.5.2). There is also Ada plug-in for Eclipse - GNATbench[24] created by AdaCore.

2.5.1 GNAT compiler

GNAT compiler is Ada compiler created by AdaCore[25]. It is part of GNU Compiler Collection (GCC). The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go. It is one of the most popular compiler systems. It is included in all Linux distributions. GNU is open source, published on GNU General Public License. GCC is divided into front end and back end. It allows to create new front end for some language and reuse existing back end.

GNAT supports Ada 2012, Ada 2005, Ada 95 and Ada 83. The front-end and run-time are written in Ada. To make compilation easier, GNAT provides `gnatmake` tool. It takes as an argument project file (.gpr) or main program file (file, which contains main procedure) and builds entire program automatically. `gnatmake` invokes GCC to perform the actual compilation. It check all dependencies contained in `.ali` files. Each invocation of GCC produce object file (`.o`) and Ada Library Information file (`.ali`). Once compilation is done, `gnatmake` invokes `gnatbind`

[23] http://libre.adacore.com/tools/gps
[24] https://www.adacore.com/gnatpro/toolsuite/gnatbench/
[25] http://www.adacore.com

tool to check consistency and generate a main program. Then `gnatlink` performs linking using binding output and all object files.

GNAT compiler is available for all most popular platforms: Windows, Linux and MacOS. AdaCore, released also GNAT cross-compiler for ARM devices. However, for now, the compilation has to be done on 32-bit Linux platform.

2.5.2 GNAT Programming Studio (GPS)

GNAT Programming Studio (GPS) is Integrated development environment for SPARK Ada. It allows to easily manage and compile SPARK Ada projects using .gpr file. GPS includes set of verification tools. More precisely GUI for setting up their options, running them and analyze results. Additionally, it enables to create plug-ins using Python and PyGTK[26]. Sireum Bakar (developed by SAnToS lab) is GPS plug-in written in Python and PyGTK. The same with other plug-ins created by AdaCore like SPARK Examiner or GNATprove.

There are two versions of GPS: free (GPL) and commercial (Pro). There are version for all most popular platforms: Windows, Linux and MacOS.

2.5.3 Ravenscar Tasking Subset

The Ravenscar Profile provides a subset of the tasking facilities of Ada95 and Ada 2005 suitable for the construction of high-integrity concurrent programs [Tea12]. RavenSPARK is SPARK subset of the Ravenscar Profile. The Ravenscar Profile is a subset of the tasking model, restricted to meet the real-time community requirements for determinism, schedulability analysis and memory-boundedness, as well as being suitable for mapping to a small and efficient run-time system that supports task synchronization and communication, and which could be certifiable to the highest integrity levels. The concurrency model promoted by the Ravenscar Profile is consistent with the use of tools that allow the static properties of

[26] http://docs.adacore.com/gps-docs/users_guide/_build/html/extending.html

programs to be verified. Potential verification techniques include information flow analysis, schedulability analysis, execution-order analysis and model checking. These techniques allow analysis of a system to be performed throughout its development life cycle, thus avoiding the common problem of finding only during system integration and testing that the design fails to meet its non-functional requirements. [AB04]

Ravenscar profile is available in SPARK 2005, but not yet in SPARK 2014[27] [AL14a]. Default profile (sequential) does not enable tasking. In other words, SPARK tools cannot analyze and reason about concurrent programs if Ravenscar profile flag is not provided.

To create a task, the task type has to be declared and task variable of this type. Ravenscar does not allows dynamic task creation. Thus, all tasks have to exists for the full lifetime of the program. [AW01] Tasks can be declared only in packages. Not in subprograms or in other tasks. [Bar13] The priority of each tasks has to be specified by `pragma Priority`. The range of available priority values is specified in the `System` package. The default range is 1 to 63. Listing 2.5.3 shows sample package with two tasks.

```
package Some_Pkg
--# own task t1 : Task1;
--#     task t2 : Task2;
is
  task type Task1
  is
    pragma Priority(10);
  end Task1;

  task type Task2
  is
    pragma Priority(9);
  end Task2;

end Some_Pkg;
```

Listing 2.4: *Sample tasks*

Declared tasks have to be implemented in the package body (listing 2.5.3).

```
package body Some_Pkg
is
  t1 : Task1;
  t2 : Task2;
```

[27] http://docs.adacore.com/spark2014-docs/html/lrm/tasks-and-synchronization.html

```
  task body Task1
  is
  begin
    loop
      -- implementation;
    end loop;
  end Task1;

  task body Task2
  is
  begin
    loop
      -- implementation;
    end loop;
  end Task2;

end Some_Pkg;
```

Listing 2.5: *Sample tasks body*

There are two ways to access variable in different tasks:

- It has to be protected object
- It has to be atomic type

Protected object encapsulate variable, in such a way that it is accessible, only through protected subprograms. This mechanism use locking, to ensure atomicity. Protected type declaration is similar to task: specification and body has to be defined. Listing 2.5.3 shows sample tasks with protected type `Integer_Store`, which enable to share Integer variable between tasks. What is important, protected type has to be declared before tasks, which will use it. Otherwise, it will be not visible for them.

```
package Some_Pkg
--# own protected Shared_Var : Integer_Store (Priority => 11);
--#     task t1 : Task1;
--#     task t2 : Task2;
is
    protected type Integer_Store
    is
        pragma Priority (11);

        function Get return Integer;
        --# global in Integer_Store;

        procedure Put(X : in Integer);
        --# global out Integer_Store;
        --# derives Integer_Store from X;
    private
```

```
        TheStoredData : Integer := 0;
    end Integer_Store;

    task type Task1
      --# global out Shared_Var;
    is
        pragma Priority(10);
    end Task1;

    task type Task2
      --# global in Shared_Var;
    is
        pragma Priority(9);
    end Task2;

end Some_Pkg;
```

Listing 2.6: *Sample tasks with protected object*

Protected type body also has to be defined in package body (listing 2.5.3).

```
package body Some_Pkg
is
    Shared_Var : Integer_Store;
    t1 : Task1;
    t2 : Task2;

    protected body Integer_Store is
        function Get return Integer
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Integer)
        --# global out TheStoredData;
        --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Integer_Store;

    task body Task1
    is
    begin
        loop
            Shared_Var.Put(5);
        end loop;
    end Task1;

    task body Task2
    is
        Local_Var : Integer;
    begin
        loop
            Local_Var := Shared_Var.Get;
        end loop;
    end Task2;

end Some_Pkg;
```

Listing 2.7: *Sample tasks with protected object body*

`Task1` is writing to `Shared_Var` and `Task2` is reading `Shared_Var`. The highest priority is assigned to protected object, to ensure atomicity during operations on it. The lowest priority is assigned to `Task2`, which is reading `Shared_Var`. Reading is usually less expensive operation than writing. Thus, to avoid starvation, `Task1` has higher priority than `Task2`. Notice, that `Shared_Var` is declared in package body, but refined in package specification.

Protected variables may not be used in proof contexts. Thus, if we try to use protected variable in proofs (pre- or postcondition), then SPARK Examiner returns following error: `Semantic Error 940 - Variable is a protected own variable. Protected variables may not be used in proof contexts.` Formal reasoning about interactions and especially temporal properties require other techniques such as model checking and lie outside the scope of SPARK [Bar13]. To preserve opportunity to use pre- and postconditions, atomic types have to be used.

To declare atomic type, `pragma Atomic` has to be used. However, there is restriction, that `pragma Atomic` cannot be applied to predefined type such as Integer. Thus, custom type has to be defined. It can be just rename of Integer. Then `pragma Atomic` can be applied on this type. Listing 2.5.3 presents previous example with atomic types instead of protected objects.

```
package Some_Pkg
--# own Shared_Var;
--#     task t1 : Task1;
--#     task t2 : Task2;
--# initializes Shared_Var;
is
    type Int32 is new Integer;

    task type Task1
      --# global out Shared_Var;
    is
        pragma Priority(10);
    end Task1;

    task type Task2
      --# global in Shared_Var;
    is
        pragma Priority(9);
    end Task2;

end Some_Pkg;
```

```
package body Some_Pkg
is
    Shared_Var : Int32 := 0;
    t1 : Task1;
    t2 : Task2;

    task body Task1
    is
    begin
        loop
            Shared_Var := 5;
        end loop;
    end Task1;

    task body Task2
    is
        Local_Var : Integer;
    begin
        loop
            Local_Var := Integer(Shared_Var);
        end loop;
    end Task2;

end Some_Pkg;
```

Listing 2.8: *Sample tasks with atomic type*

It is important to mention, that `pragma Atomic` does not guaranty atomicity. In most cases, atomic types should not be used for tasking. Instead, protected types should be used. When an object is declared as atomic, it just means that it will be read from or written to memory atomically. The compiler will not generate atomic instructions or memory barriers when accessing to that object. `pragma Atomic` force compiler only to:

- check if architecture guarantees atomic memory loads and stores,
- disallow some compiler optimizations, like reordering or suppressing redundant accesses to the object

Another important thing in tasking is Time library: `Ada.Real_Time`. It allows to run task periodically, using `delay until` statement, which suspends task until specified time. To use `delay` in the task, it has to be declared in `declare` annotation: `--# declare delay;` [Bar13].

Details about tasking in SPARK are well described in Chapter 8 of [Bar13]. The "Guide for the use of the Ada Ravenscar profile in high integrity systems" [AB04] and the official

Ravenscar Profile documentation (which includes examples) [Tea12] is another good source. The limitations of Tasking in SPARK are reviewed in Audsley's and Welllings' paper [AW01].

2.6 SPARK Ada Verification

The goal of software verification is to assure software correctness and lack of errors. There are two types of verification:

- dynamic - performed during the execution of software, e.g. unit tests
- static - achieved by formal methods, mathematical calculations and logical evaluations

Dynamic verification starts with a set of possible test cases, simulates the system on each input, and observes the behavior. In general, it does not cover all possible executions. On the other hand, static verification establishes correctness for all possible execution sequences. Static and dynamic verification can be mixed, e.g. by generating test cases with static verification tools and then proving correctness with unit tests during runtime [DRH07].

Techniques for Static Verification:

- Formal verification: prove mathematically that the program is correct - this can be difficult for large programs.
- Correctness by construction: follow a well- defined methodology for constructing programs.
- Model checking: enumerate all possible executions and states, and check each state for correctness.

SPARK consists of a verification tool-set:

- SPARKMake - generates index file (.idx) and meta file (.smf)

[28] http://docs.adacore.com/sparkdocsdocs/Examiner_UM.htm

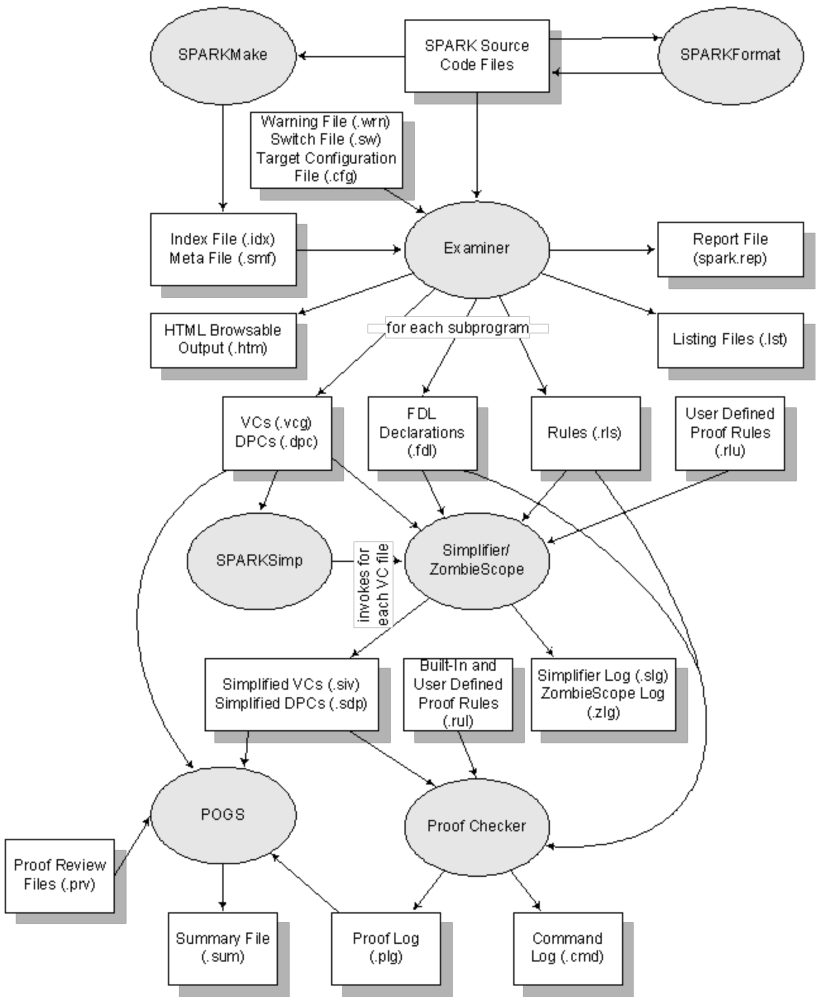

Figure 2.6: *Relationship of the Examiner and Proof Tools*[28].

- Examiner - check syntax, generates Verification Conditions (VCs) and Dead Path Conjectures (DPCs), and discharge (prove) them
- Simplifier - simplify and discharge VCs, which are not discharged by Examiner
- ZombieScope - find dead paths
- ViCToR - translate VCs and DPCs to format acceptable by SMT solver and prove correctness using specified SMT solver
- SPARKSimp - runs Simplifier or/and ZombieScope

- POGS - produces verification report
- Proof Checker - discharge VCs or DPCs not discharged by Examiner and Simplifier

Relationships between tools and verification flow is presented on figure 2.6. SPARK proof tools use FDL as the modeling language.

2.6.1 SPARK Examiner

The main SPARK verification tool is Examiner. It supports several levels of analysis:

- checking of SPARK language syntactic and static semantic rules
- data flow analysis
- data and information flow analysis
- formal program verification via generation of verification conditions
- proof of absence of run-time errors
- dead path analysis

There is also an option to make the Examiner perform syntax checks only. Using this option on a source file does not require access to any other units on which the file depends, so files can be syntax checked on an individual basis. This allows any syntax errors to be corrected before the file is included in a complex examination. This option must only be used as a pre-processor: the absence of syntax errors does NOT indicate that the source text is a legal SPARK program. [Tea11b] [THIS PART IS COPY AND PASTE FROM Examiner doc - is it ok?]

[Put here some examples? E.g.: method without contract, examine, add specification, pass Examiner.]

Examiner can perform data and information analysis of Ravenscar programs in exactly the same manner as for sequential programs [Tea12]. Unfortunately it does not allow protected objects in proof annotations (pre- and post-conditions) as mentioned in section 2.5.3.

When some parts of the system are written in full Ada (with non-valid SPARK constructs), then Examiner returns error. Ada parts can be excluded from Examiner analysis using `--# hide` annotation. Then, only a warning is returned by Examiner: `10 - The body of subprogram Main is hidden - hidden text is ignored by the Examiner.`

Examiner use SPARK index file (.idx) - generated by `SPARKMake` tool - to locate files necessary for verification. [Bar13]

Examiner can be used with `spark` command and appropriate flags described in Examiner Manual [Tea11b].

To use Examiner in GNAT Programming Studio:

- Run SPARK Make: right click on project / SPARK / SPARK Make (figure 2.7)
- Set SPARK index file (to spark.idx generated by SPARKMake) (figure 2.8)
- (optionally) set configuration file (e.g. Standard.ads)
- Choose appropriate version of SPARK (95 or 2005)
- Choose mode: Sequential (for single tasking programs) or Ravenscar (for multitasking programs)

To generate verification conditions (VCs), the `-vcg` switch has to be used. It can be set in GNAT Programming Studio (Project / Edit project properties / Switches / Examiner / Generate VCs). In addition to verification conditions, Examiner can check dead path conjectures (DPCs). It checks, whether all of the program is useful. To generate dead path conjectures, the `-dpc` switch has to be used. It can be also set in GNAT Programming Studio (Project / Edit project properties / Switches / Examiner / Generate DPCs).

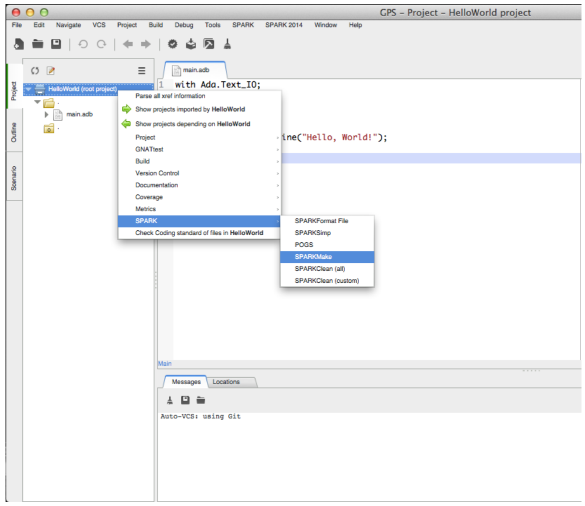

Figure 2.7: *Run SPARK Make*

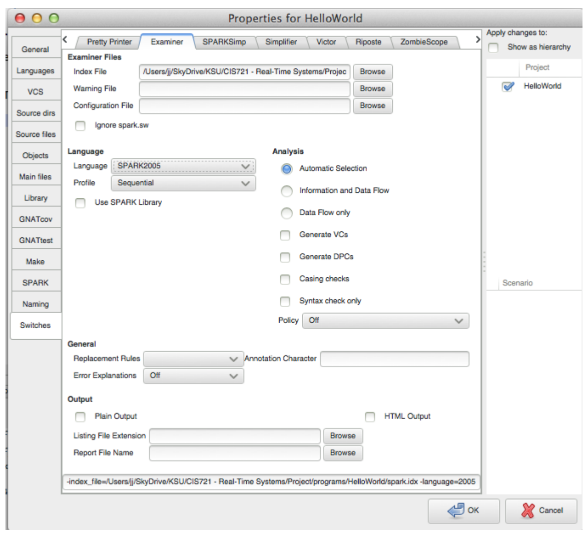

Figure 2.8: *Examiner Properties*

Flow analysis

There are two types of flow analysis:

- Data flow analysis:
  - Checks input/output behavior of parameters and variables.

  - Checks initialization of variables.
  - Checks that changed and imported variables are used later (possibly as output variables).

- Information flow analysis - verifies interdependencies between variables.

In data flow analysis, Examiner checks if input parameters are not modified, but used at least once (in at least one branch of program). In the same factor, output parameters cannot be read (before initialization) and has to be initialized (in all branches of program). Input/output parameters has to be both read and write (changed). In similar way, Examiner verify the global variables (specified in annotations). Functions can use only input parameters and can only read global variables. Therefore functions do not have side effects.

Global variables defined in package body (thus private) has to be declared by `--# own` annotation in package specification. If variable is also initialized, `--# initializes` annotation has to be used. In Ada, to use package in another package, **with** clause has to be used. In SPARK Ada, additionally `--# inherits` annotation has to be specified.

In information flow analysis, dependencies between variables are analyzed. These dependencies are specified by `--# derives` annotation.

Verification conditions

To generate verification conditions, two kinds of annotations are relevant for Examiner:

- preconditions: `--# pre`

- postconditions: `--# post`

Notion of pre- and postconditions represents Hoare logic. More precisely, Hoare triple:

$$\{P\}C\{Q\} \tag{2.1}$$

P and Q are assertions. C is a command (action) performed between them. P is precondition and Q is post-condition.

Additionally, assertions (`--# assert`) and checks (`--# check`) can be specified in procedure body. Then additional verification conditions are generated.

Functions does not have side effects (as stated in 2.6.1), thus only precondition can be applied. However, there is annotation `--# return`, which specify function return value.

Verification conditions are generated depended on number of paths in subprogram. Analysis is performed backwards, in other words: we start from post-conditions and consider what must holds before. Flow analysis is well described in chapter 11 of [Bar13].

If preconditions are not present, then the formula expresses that the post-condition holds always.

2.6.2 SPARK Simplifier

Simplifier, simplify verification conditions (VCs) generated by Examiner. It can also discharge (prove correctness) of those VCs, which are not proved by Examiner. [Tea11c] It takes as input `.vcg` files, `.fdl` files for its data declarations and - if available - proof-rule files (`.rls`, `.rlu`). Then it generates `.siv` files (simplified VCs) and `.slg` files (details about simplification, which has been made).

2.6.3 ZombieScope

ZombieScope is a SPARK tool, that analyze SPARK code to find dead paths, i.e. paths through the code that can never be executed. Program, which contains dead paths may not necessarily be incorrect, but a dead path is an indication of a potential code issue.

ZombieScope reads `.dpc` files generated by the Examiner. In order to generate dead path conjectures, `-dpc` flag has to be used or 'Generate DPCs' option has to be checked in Examiner options, in GPS. It reads also `.fdl` files for its data declarations and the `.rls` file for proof-rules if present. ZombieScope generates two output files: `.sdp` file (dead path summary) and `.zlg` file (details about underlying contradiction search performed). ZombieScope is invoked by SPARKSimp by default and the summary file generated by POGS includes information about the dead path analysis.

2.6.4 ViCToR

ViCToR is a tool to translate SPARK verification conditions (VCs), as generated by the Examiner, into SMT-LIB (file format used to communicate with SMT solvers). [Tea] SMT (Satisfiability Modulo Theories) solver is a tool for verification and proving the correctness of programs. ViCToR is integrated with SPARKSimp and POGS. To invoke ViCToR from SPARKSimp, flag `-victor` has to be used.

2.6.5 Proof Checker

Proof Checker is advanced verification tool, which require considerable experience in verification of SPARK programs. It is interactive program, which enables the user to direct the Checker to explore the use of various strategies and rules on the condition to be proved. Proof Checker can keep a log of the progress of a proof in `plg` file. It also keep command record in `cmd` file. More details about Proof Checker can be found in chapter 12 of [Bar13].

2.6.6 SPARKSimp Utility

SPARKSimp is a simple "make" style tool for the SPARK analysis tools. Currently, it supports the Simplifier, ZombieScope and ViCToR. It applies the Simplifier (and ViCToR, if requested) to all `.vcg` files and ZombieScope to all `.dpc` files, which it finds in a directory tree. [Tea10]

2.6.7 Proof Obligation Summarizer (POGS)

The Proof ObliGation Summarizer tool (POGS) reads and understands the structure of the verification conditions (`.vcg` files), their simplified version (`.siv` files) and dead path conjectures (`.dpc` files). It reports the status of proofs and dead path analyses in a human-readable, text form. [Tea11a]

2.6.8 AUnit

AUnit is Unit Test Framework for Ada language. It can be also applied for verify SPARK Ada programs. It was created based on Java JUnit (created by Kent Beck, Erich Gamma) and C++ CppUnit (created by M. Feathers, J. Lacoste, E. Sommerlade, B. Lepilleur, B. Bakker, S. Robbins) unit test frameworks [Ada14]. Similar like mentioned frameworks it enables simple test cases testing, fixtures, suites and provides reporting [Fal14]. As mentioned at the beginning of section 2.6 it is used mainly for dynamic verification.

GNAT Programming Studio can generate test cases skeleton for all subprograms. It can be generated using Tools -> GNATtest -> Generate unit test setup. This generator creates new project with AUnit tests. Project for which tests are generated is referenced in new generated project. In order to run tests, the test project has to be opened in GNAT Programming Studio. The project is created in `[project_dir]/gnattest/harness/test_[proj_name].gpr`. It generates empty (not implemented) test for each subprogram in project. To add/edit/remove tests or rename names, three files has to be edited:

- `[some_package]-test_data-tests.ads`
- `[some_package]-test_data-tests.adb`
- `[some_package]-test_data-tests-suite.adb`

Test has to be declared in `[some_package]-test_data-tests.ads` and implemented in `[some_package]-test_data-tests.adb`. Then it has to be added to test suite in `[some_package]-test_data-tests-suite.adb` file.

Tests can be also created manually. Then the AUnit distribution has to be referenced in project file and all test cases (and suits) has to be implemented by hand.

2.6.9 Sireum Bakar

Sireum[29] is a long-term research conducted by SAnToS lab at Kansas State Univeristy. Its goal is to develop an over-arching software analysis platform that incorporates various static analysis techniques such as data-flow framework, model checking, symbolic execution, abstract interpretation, and deductive reasoning techniques (e.g., using weakest precondition calculation). It can be used to build various kinds of software static analyzers for different kinds of properties.

It uses the Pilar language [SC12] as intermediate representation. Any language which can be translated to Pilar can be analyzed by Sireum. For now, there is translator for SPARK and Java.

Bakar is a toolset for analyzing SPARK Ada programs (Bakar means "spark" in Indonesian). Sireum Bakar currently includes:

- Kiasan - functional behaviors verification tool
- Alir - information flow analysis tool

Sireum distribution is available for Windows (32-bit, 64-bit), Linux (32-bit, 64-bit) and MacOS (64-bit). It can be downloaded from `http://www.sireum.org/`.

[29] http://www.sireum.org/

Bakar Kiasan

Bakar Kiasan [BHR+11] is a fully automated tool for verifying functional behaviors of SPARK programs specified as software contract (Kiasan means "symbolic" in Indonesian). Kiasan use symbolic execution technique. It provides various helpful feedback including generation of counter example for contract refutation, test cases for an evidence of contract satisfaction, verification reports, visual graphs illustrating pre/post states of SPARK procedures/functions, etc. It is much easier for hazard analysis than e.g. analysis of `.vcg` files generated by SPARK Examiner.

There exists Kiasan Plug-in for GNAT Programming Studio (GPS). Version 1, for GPS 5, supports SPARK 2005. Version 2, for GPS 6, which supports 2014 is under development. Both plug-ins are created by author of this thesis in Python and PyGTK. There is also plug-in for Eclipse, but only for SPARK 2005 programs.

Bakar Kiasan does not support Ravenscar profile. Thus, it can be used only for sequential programs verification. Figure 2.9 depicts sample Kiasan analysis result. Kiasan window has two parts: list of units (packages and subprograms) and analysis cases with pre and post states. Every unit has associated statistics:

- T# - Test cases (expected behavior)
- E# - Exception cases (unexpected behavior)
- Instruction coverage - amount of code covered by Kiasan analysis
- Branch coverage - number of branches covered by analysis (0% in 100% instruction coverage means, that there is no branches in analyzed unit)
- Time in which analysis was performed

After double click on some unit, code which is executed during execution of this unit is highlighted. Additionally below the list of units, there is a combo box which contains all test cases associated with selected (by double click) unit. Once, some case is selected, code coverage equivalent to this test case is highlighted. Additionally, below combo box, there are states of unit execution. On the left hand side, there is pre-state, and on the right hand side there post-state of analysis. Variables with red font color, in post-state, are those which are changed in result of unit execution. The new created variables (during unit execution) are blue, but there are not present in figure 2.9.

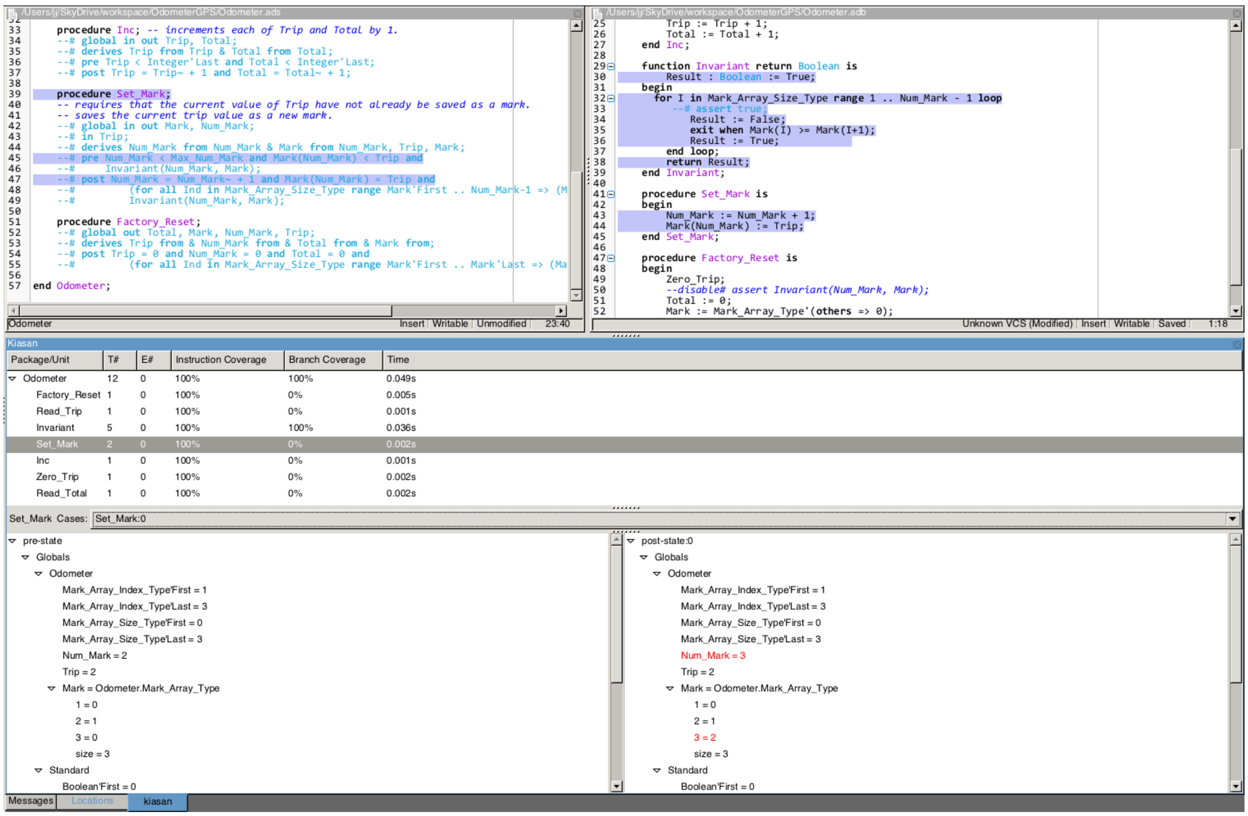

Figure 2.9: *Bakar Kiasan report*

Bakar Kiasan is useful especially, for solving verification issues. It can generate counter examples, which help to fix the code. [add screenshot with error case and discuss result?]

Bakar Alir

Alir is an information flow analysis tool for reasoning about SPARK's derive clauses/information flow (Alir means "flow" in Indonesian). Alir visualizes information flows to ease engineers in understanding information dependencies crucial for specifying and verifying SPARK's derive clauses. It provides various configurable intra-procedural and inter-procedural analyses. The inter-procedural analyses are control flow analysis, reaching definition analysis and data dependence analysis. The inter-procedural analysis in Alir includes building, System Dependence Graph (SDG), slicing and chopping on SDG. [Thi11]

2.6.10 GNATprove

GNATprove [30] is a formal verification tool for SPARK 2014 programs. It is based on the GNAT compiler. GNATprove interprets SPARK Ada annotations exactly like they are interpreted at run time during tests. It can prove that subprograms respect their contracts, expressed as preconditions and postconditions in the syntax of Ada 2012. The tool automatically discovers the subset of subprograms which can be formally analyzed. GNATprove is currently available for x86 linux, x86 windows and x86-64 linux.

GNATprove consists of two distinct analyses, flow analysis and proof. Flow analysis checks the correctness of aspects related to data flow (`Global`, `Depends`, `Abstract_State`, `Initializes`, and refinement versions of these), and verifies the initialization of variables. Proof verifies the absence of runtime errors and the correctness of assertions such as `Pre` and `Post` aspects. Using the switch `--mode=<mode>`, whose possible values are `flow`, `prove` and `all`, only one or both of these analyses can be performed (`all` is the default). [AL14b]

GNATprove use Alt-Ergo prover for verification.

[Add more details? Some example like in Kiasan section?]

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

2.7 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. Prototyping Embedded Systems using AADL lasts for a few years [CB09]. There are already existing tools, which performs code generation based on AADL:

- Ocarina
- Ramses

2.7.1 Ocarina

Ocarina [LZPH09] is a tool suite, which contains plug-ins for code generation, model checking and analysis. The code generation plug-in generates code from an AADL architecture model to an Ada or C 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. Ocarina is written in Ada.

There is plug-in for OSATE (see section 2.3.1), which enables code generation. Example AADL models, suitable for being an input of Ocarina are available on github repository: `https://github.com/yoogx/polyorb-hi-ada/tree/master/examples/aadlv2`.

Since mid-2009, Telecom ParisTech is no longer involved in Ocarina, and is developing another AADL tool-chain, based on Eclipse, codenamed RAMSES [Hug13].

[Include some examples and generated code? E.g. prod-cons example?]

2.7.2 RAMSES

RAMSES (Refinement of AADL Models for Synthesis of Embedded Systems) is a model transformation and code generation tool. It is written in Java. RAMSES produces C code, but does not generate Ada. It simplify AADL models, in order to generate C code. Simplified

AADL models contain behavior annex subclauses. RAMSES can be used as OSATE plug-in or standalone application.

[I didn't find much about RAMSES online...]

Chapter 3

PCA Pump

Patient Controlled Analgesia (PCA) pump is a medical device, which allows the patient to self-administer small doses of narcotics (usually Morphine, Dilaudid, Demerol, or Fentanyl). PCA pumps are commonly used after surgery to provide a more effective method of pain control than periodic injections of narcotics. A continuous infusion (called a basal rate) permits the patient to receive a continuous infusion of pain medication. There is no need for a clinician to administer it. Patient can also request additional boluses, but only in specified intervals. It prevents from over infusion. In addition to basal and patient bolus, clinician can also request bolus called clinician bolus or square bolus.

Figure 3.1: *Patient Controlled Analgesia (PCA) pump*

Figure 3.1 shows LifeCare PCA pump. On the left hand side, there is drug reservoir. On the right - clinician panel, which allows to control the pump. Figure 3.2 shows PCA

Pump, made by company Alaris.

Figure 3.2: *Alaris Pump*

PCA Pump is safety-critical device which works in standard process control loop depicted in the figure 3.3. The controller obtains information about (observes) the process state from measured variables (feedback) and uses this information to initiate action by manipulating controlled variables to keep the process operating within predefined limits or set points (the goal) despite disturbances to the process. Such as different air pressure or device position (gravity impact). In general, the maintenance of any open-system hierarchy (either biological or man-made) will require a set of processes in which there is communication of information for regulation or control. [Lev12]

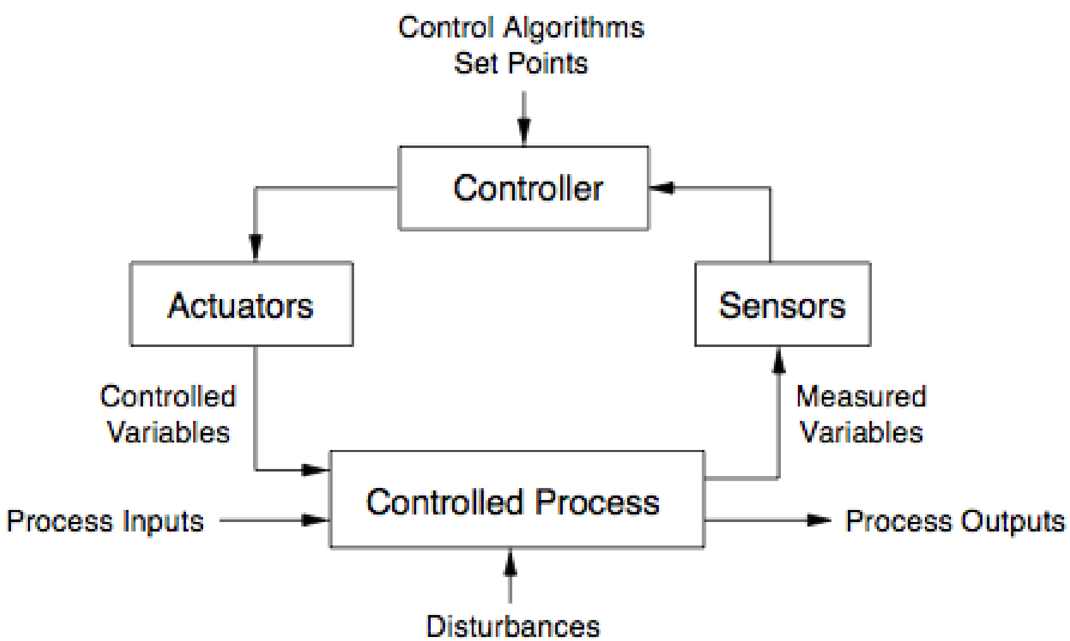

Figure 3.3: *Standard Process Control Loop.*

PCA Pump actuator is motor, which pump drug to the patient's vein. Controlled process is dosing the drug. Sensors measure amount of dosed drug. They might be used for double-check if ordered (by controller) amount of drug was appropriately delivered. Sometimes there might be some distrubances caused by mechanical issues and environmental conditions. Controller issues appropriate actions based on informations from sensors and clinician or patient's commands. High level overview of PCA Pump is depicted in the figure 3.4.

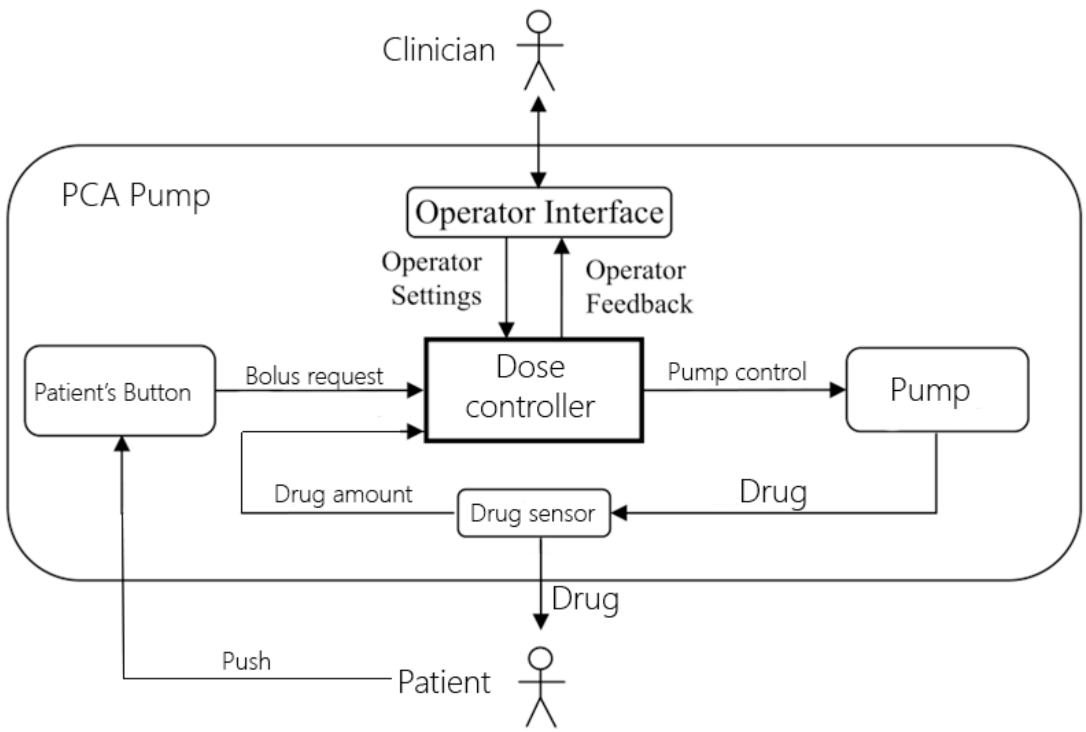

Figure 3.4: *PCA Pump system*

One of the hazards of using PCA pumps, is that there is inadequate monitoring of patient's levels of oxygen and carbon dioxide. Nursing staff on general medical units typically track respiration rate and other vital signs every four hours, which is not enough. There should be a way to monitor levels continuously. Additionally, it can be hard to tell if a person's breathing rate is dangerously low in certain circumstances. There are cases, where lack of monitoring carbon dioxide level caused death.[1]

Another hazard is human mistake. For example, there is a case when nurse used a 5 mg/mL morphine cassette because a 1 mg/mL cassette was not available, but she programmed PCA Pump like for 1 mg/mL concentration. In addition to lack of monitoring of the pulse, patient died.[2]

As mentioned in chapter 2, the solution to that problem is medical devices interoperability. In addition, less human error-prone device is needed. It can be assured by using more than one system for their detection.

3.1 PCA Pump Requirements Document

Requirements of "Open Source PCA Pump" [LHC13] are captured in "Integrated Clinical Environment Patient-Controlled Analgesia Infusion Pump System Requirements" document [Lar14] created by Brian Larson. It is formalized set of capabilities, which Open PCA Pump should have, based on consultations with domain experts, FDA and Brian Larson's expertise gained while he was working in the medical device industry.

Conceptual model of Open PCA pump is depicted in the figure 3.5. As mentioned earlier, the pump is connected to ICE so it may be integrated with ICE apps and displays. The interface must provide prescription and patient information, current status to be displayed remotely on a supervisor user interface, and a means to stop infusing upon human command, or determination of an ICE app. Such an ICE app could monitor a patient's blood oxygenation and pulse rate, stopping the pump if depressed respiratory function is indicated. [Lar14]

Additionally, it cooperates with Drug Library, which contains information about drugs and its properties (like concentration). Data needed for pump operation, are captured on

[1]http://abcnews.go.com/Health/parents-warn-pca-pumps-daughters-death/story?id=16796805
[2]http://webmm.ahrq.gov/case.aspx?caseID=291

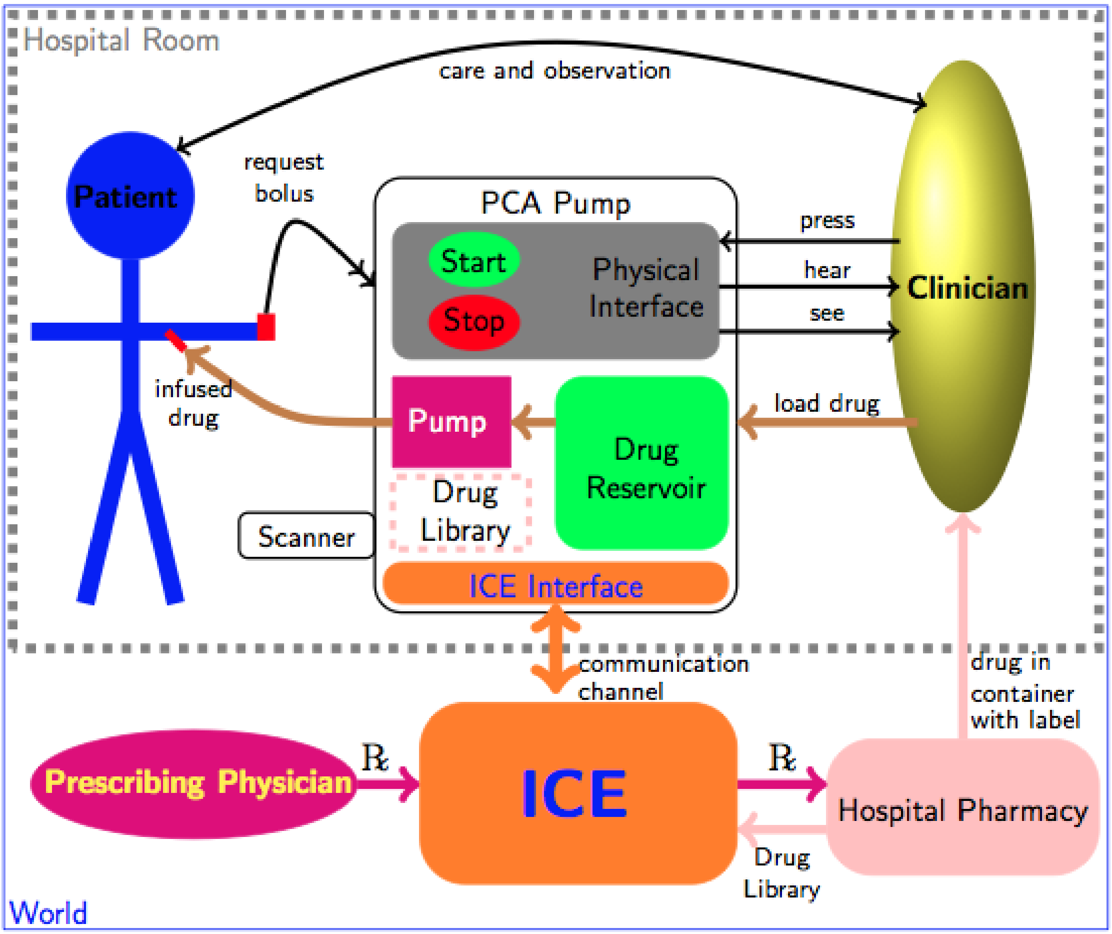

Figure 3.5: *Open PCA Pump concept*

electronic prescription, which contains:

- Patient's name
- Drug name
- Drug code
- Drug concentration
- Initial volume of drug in the vial
- Basal flow rate - the rate of continuous infusion

- Volume to be infused (VTBI) on patient's request
- Maximum amount of drug allowed per hour
- Minimum time between patient boluses
- Date, in which prescription has been filled
- Prescribing physician's name
- Pharmacist name

Pain medication is prescribed by a licensed physician, which is dispensed by the hospital's pharmacy. The drug is placed into a vial labeled with the name of the drug, its concentration, the prescription, and the intended patient. A clinician loads the drug into the pump, and attaches it to the patient. The pump infuses a prescribed basal flow rate which may be augmented by a patient-requested bolus or a clinician-requested bolus. This allows additional pain medication in response to patient need within safe limits. [Lar14]

Prescription captures all data needed for basal infusion and patient requested boluses (referred as bolus). In addition to that, Open PCA Pump allows Clinician Requested Bolus (refereed as square bolus). In order to do that, clinician has to enter the time (through PCA Pump panel) in which VTBI, specified in prescription, will be infused.

There can occur situations in which the maximum drug amount infused may exceed the allowed limit. E.g. when clinician issues too many square boluses. In such case, pump is switched to Keep Vein Open (KVO) mode, which has 1 ml/hr drug rate. Pump switches to KVO rate also when ICE interface request it. It may happen e.g. if patient's oxygen level is low. To recover from KVO state, pump has to be restarted by clinician in order to continue operation. In Summary, Open PCA Pump has following modes:

- Stopped

- Basal rate
- Patient's bolus (bolus)
- Clinician bolus (square bolus)
- Keep Vein Open (KVO)

There are also other scenarios, which are captured by Requirements Document [Lar14], like scanner to enable automatic entry of patient's and prescription data, occlusion detection, hardware errors alarms etc. Detailed overview of Open PCA Pump Requirements can be found in [LHC13].

[MORE DETAILS ABOUT PUMP?] [ADD STATE MACHINE IMAGE, LIKE IN UMINN REQ COD?]

3.2 PCA Pump AADL/BLESS Models

In addition to PCA Pump Requirements Document [Lar14], Brian Larson created AADL model with formal behavioral specifications written in his BLESS framework. AADL model, graphical representation is depicted on figure 3.6.

AADL model captures structure of device. BLESS - its behavior. Listing 3.2 shows `Rate_Controller` thread from `PCA_Operation` component with BLESS assertions in thread declaration and BLESS behavioral description in thread implementation. The thread declaration contains input and output ports. In addition to some of them, BLESS assertions are present. Assertions are defined in BLESS annex in thread implementation. In addition to assertions, states and transitions defined in thread implementation can potentially be translated into working SPARK Ada program. Presence of timing properties in states and transitions makes translation extremely difficult, thus there are omitted in this thesis and only assertions are considered. [TRUNCATE CODE LISTING?]

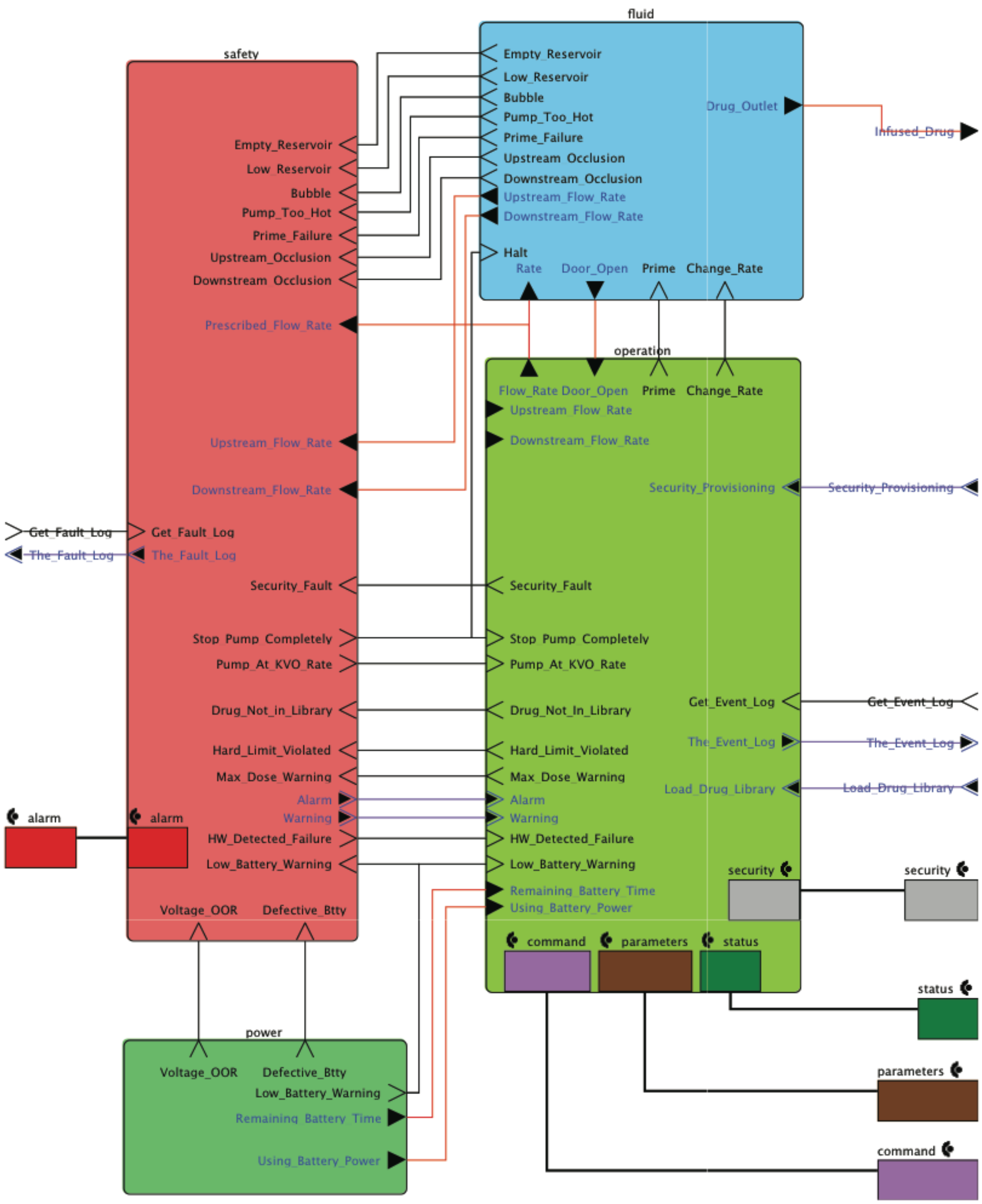

Figure 3.6: *Open PCA Pump AADL model*

```
thread Rate_Controller
features
    Infusion_Flow_Rate: out data port PCA_Types::Flow_Rate
      {BLESS::Assertion => "<<:=PUMP_RATE()>>";};
    System_Status: out event data port PCA_Types::Status_Type;
    Begin_Infusion: in event port
      {BLESS::Assertion => "<<Rx_APPROVED()>>";};
    Begin_Priming: in event port;
    End_Priming: in event port;
    Halt_Infusion: in event port;
    Square_Bolus_Rate: in data port PCA_Types::Flow_Rate
      {BLESS::Assertion => "<<:=SQUARE_BOLUS_RATE>>";};
    Patient_Bolus_Rate: in data port PCA_Types::Flow_Rate
      {BLESS::Assertion => "<<:=PATIENT_BOLUS_RATE>>";};
    Basal_Rate: in data port PCA_Types::Flow_Rate
      {BLESS::Assertion => "<<:=BASAL_RATE>>";};
    VTBI: in data port PCA_Types::Drug_Volume
      {BLESS::Assertion => "<<:=VTBI>>";};
    HW_Detected_Failure: in event port;
    Stop_Pump_Completely: in event port;
    Pump_At_KVO_Rate: in event port;
    Alarm : in event data port PCA_Types::Alarm_Type;
    Warning : in event data port PCA_Types::Warning_Type;
    Patient_Request_Not_Too_Soon: in event port
      {BLESS::Assertion => "<<:=PATIENT_REQUEST_NOT_TOO_SOON(now)>>";};
    Door_Open: in data port Base_Types::Boolean;
    Pause_Infusion: in event port;
    Resume_Infusion: in event port;
    CP_Clinician_Request_Bolus: in event port;
    CP_Bolus_Duration: in event data port ICE_Types::Minute;
    Near_Max_Drug_Per_Hour: in event port  --near maximum drug infused in any hour
      {BLESS::Assertion => "<<PATIENT_NEAR_MAX_DRUG_PER_HOUR()>>";};
    Over_Max_Drug_Per_Hour: in event port  --over maximum drug infused in any hour
      {BLESS::Assertion => "<<PATIENT_OVER_MAX_DRUG_PER_HOUR()>>";};
    ICE_Stop_Pump: in event port;
  properties
    Thread_Properties::Dispatch_Protocol => Aperiodic;
end Rate_Controller;

thread implementation Rate_Controller.imp
annex BLESS
{**
assert
<<HALT : :(la=SafetyPumpStop) or (la=StopButton) or (la=EndPriming)>>  --pump at 0 if stop button, or
   safety architecture says, or done priming
<<KVO_RATE : :(la=KVOcommand) or (la=KVOalarm) or (la=TooMuchJuice)>>  --pump at KVO rate when commanded,
   some alarms, or excedded hourly limit
<<PB_RATE : :la=PatientButton>>  --patient button pressed, and allowed
<<CCB_RATE : :(la=StartSquareBolus) or (la=ResumeSquareBolus)>>  --clinician-commanded bolus start or
   resumption after patient bolus
<<PRIME_RATE : :la=StartPriming>>  --priming pump
<<BASAL_RATE : :(la=StartButton) or (la=ResumeBasal) or (la=SquareBolusDone)>>  --regular infusion
<<PUMP_RATE : :=
  (HALT()) -> 0,                                      --no flow
  (KVO_RATE()) -> PCA_Properties::KVO_Rate,           --KVO rate
  (PB_RATE()) -> Patient_Bolus_Rate,  --maximum infusion upon patient request
  (CCB_RATE()) -> Square_Bolus_Rate,                  --square bolus rate=VTBI/duration, from data port
  (PRIME_RATE()) -> PCA_Properties::Prime_Rate,       --pump priming
  (BASAL_RATE()) -> Basal_Rate                        --basal rate, from data port
>>
invariant <<true>>
variables
  --time of last action
  tla :BLESS_Types::Time := 0;
```

```
  la :   --last action
    enumeration (
      SafetyStopPump,   --safety architecture found a problem
      StopButton,       --clinician pressed stop button
      KVOcommand,       --from control panel (clinician) or ICE (app) to pump Keep-vein-open rate
      KVOalarm,         --some alarms should pump at KVO rate
      TooMuchJuice,     --exceeded max drug per hour, pump at KVO until prescription and patient are re-
   authenticated
      PatientButton,    --patient requested drug
      ResumeSquareBolus,--infusion of VTBI finished, resume clinician-commanded bolus
      ResumeBasal,      --infusion of VTBI finished, resume basal-rate
      StartSquareBolus, --begin clinician-commanded bolus
      SquareBolusDone,  --infusion of VTBI finished
      StartPriming,     --begin pump/line priming, pressed "prime" button
      EndPriming,       --end priming, pressed "prime" button again, or time-out
      StartButton       --start pumping at basal rate
    );
  pb_duration :BLESS_Types::Time  --patient button duration = VTBI/Patient_Bolus_Rate
    <<PB_DURATION : :pb_duration=(VTBI/Patient_Bolus_Rate)>>;
states
  PowerOn : initial state;        --power-on
  WaitForRx : complete state;   --wait for valid prescription
  CheckPBR : state  --check Patient_Bolus_Rate is positive
    <<Rx_APPROVED()>>;
  RxApproved : complete state   --prescription verified
    <<Rx_APPROVED() and PB_DURATION()>>;
  Priming : complete state      --priming the pump, 1 ml in 6 sec
    <<(la=StartPriming) and (Infusion_Flow_Rate@now = PCA_Properties::Prime_Rate) and PB_DURATION()>>;
  WaitForStart : complete state   --wait for clinician to press 'start' button
    <<HALT() and (Infusion_Flow_Rate@now=0) and PB_DURATION()>>;
  PumpBasalRate : complete state  --pumping at basal rate
    <<((la=StartButton) or (la=ResumeBasal)) and (Infusion_Flow_Rate@now=Basal_Rate@now) and PB_DURATION()
   >>;
  PumpPatientButtonVTBI : complete state  --pumping patient-requested bolus
    <<(la=PatientButton) and PB_DURATION()
      and (Infusion_Flow_Rate@now=Patient_Bolus_Rate)>>;
  PumpCCBRate : complete state    --pumping at clinician-commanded bolus rate
    <<((la=StartSquareBolus) or (la=ResumeSquareBolus)) and (Infusion_Flow_Rate@now=Square_Bolus_Rate@now)
   and PB_DURATION()>>;
  PumpKVORate : complete state    --pumping at keep-vein-open rate
    <<((la=KVOcommand) or (la=KVOalarm) or (la=TooMuchJuice))  and PB_DURATION()
      and (Infusion_Flow_Rate@now=PCA_Properties::KVO_Rate)>>;
  PumpingSuspended : complete state  --clinician pressed 'stop' button
    <<((la=StopButton) or (la=SafetyStopPump)) and (Infusion_Flow_Rate@now=0)>>;
  Crash : final state;    --abnormal termination
  Done : final state    --normal termination
    <<Infusion_Flow_Rate@now=0>>;
transitions
--wait for valid prescription
  go : PowerOn-[ true ]->WaitForRx{};
--prescription validated
  rxo : WaitForRx-[ on dispatch Begin_Infusion ]-> CheckPBR{};
  pbr0 : CheckPBR-[ Patient_Bolus_Rate<=0 ]->Crash{}; --bad Patient_Bolus_Rate
  pbrok : CheckPBR-[ Patient_Bolus_Rate>0 ]->RxApproved
    {<<Rx_APPROVED() and (Patient_Bolus_Rate>0)>>  --likely will change from logic variable to predicate
   Rx_APPROVED()
      pb_duration := VTBI/Patient_Bolus_Rate  --calculate patient bolus duration
      --note division without knowing divsor is non-zero; should generate additional proof obligations for
   assignment using division
    <<Rx_APPROVED() and PB_DURATION()>>};
--clinician press 'prime' button
  rxpri : RxApproved-[ on dispatch Begin_Priming ]-> Priming
    {
    la :=StartPriming
        <<Begin_Priming@now and Rx_APPROVED() and (la = StartPriming) and PB_DURATION()>>
```

```
    ;
    Infusion_Flow_Rate!(PCA_Properties::Prime_Rate)  --infuse at prime rate
        <<(la = StartPriming) and Rx_APPROVED() and PB_DURATION() and
          (Infusion_Flow_Rate@now=PCA_Properties::Prime_Rate)>>
    };
--priming done, wait for start
  prd: Priming-[ on dispatch End_Priming or timeout (Begin_Priming) PCA_Properties::Prime_Time sec]->
   WaitForStart
    {
    la:=EndPriming
      <<HALT() and PB_DURATION()>>  --and Begin_Priming timed out
    ;
    Infusion_Flow_Rate!(0)   --stop priming flow
      <<HALT() and (Infusion_Flow_Rate@now=0) and PB_DURATION()>>
    };
--prime again
  pri: WaitForStart-[ on dispatch Begin_Priming ]-> Priming
    {
    la:=StartPriming
        <<Begin_Priming@now and PB_DURATION() and PRIME_RATE()>>
    ;
    Infusion_Flow_Rate!(PCA_Properties::Prime_Rate)  --infuse at prime rate
        <<PRIME_RATE() and PB_DURATION() and
          (Infusion_Flow_Rate@now=PCA_Properties::Prime_Rate)>>
   };
--clinician press 'start' button after priming
  sap: WaitForStart-[ on dispatch Begin_Infusion ]-> PumpBasalRate  --start after priming
    {
    la:=StartButton
      <<(la=StartButton) and Begin_Infusion@now  and PB_DURATION()>>
    ;
    Infusion_Flow_Rate!(Basal_Rate)   --infuse at basal rate
      <<(la=StartButton) and (Infusion_Flow_Rate@now=Basal_Rate@now) and PB_DURATION()>>
    };
--Patient_Request_Bolus during basal rate infusion
  pump_basal_rate:
  PumpBasalRate-[ on dispatch Patient_Request_Not_Too_Soon]-> PumpPatientButtonVTBI
    {
    la := PatientButton
      <<(la=PatientButton) and Patient_Request_Bolus@now and PB_DURATION()>>
    ;
    Infusion_Flow_Rate!(Patient_Bolus_Rate)   --infuse at patient button rate
      <<(la=PatientButton) and PB_DURATION()
        and (Infusion_Flow_Rate@now=Patient_Bolus_Rate)>>
    };  --end of pump_basal_rate
--VTBI delivered
  vtbi_delivered:
  PumpPatientButtonVTBI -[ on dispatch timeout (Infusion_Flow_Rate) pb_duration ms ]-> PumpBasalRate
    {
    la:=ResumeBasal
    ;
    <<(la=ResumeBasal) and PB_DURATION()>>  --and timeout of patient button duration
    Infusion_Flow_Rate!(Basal_Rate)   --infuse at basal rate
      <<(la=ResumeBasal) and (Infusion_Flow_Rate@now=Basal_Rate@now) and PB_DURATION()>>
    };  --end of vtbi_delivered
**};
end Rate_Controller.imp;
```

Listing 3.1: *`Rate_Controller` thread from `PCA_Operation` component with BLESS assertions*

3.3 BeagleBoard-xM

For Research and MDCF purposes, BeagleBoard-xM (an open-source hardware single-board computer produced by Texas Instruments), has been chosen as hardware platform for PCA pump prototyping.

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). Moreover there is PWM support, which enables control of pump actuator.

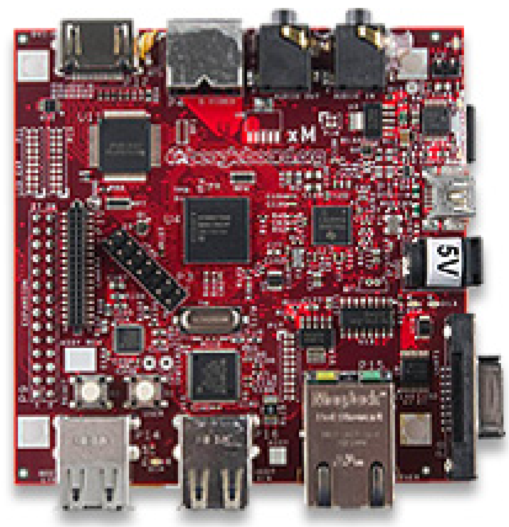

Figure 3.7: *BeagleBoard-xM*

Pulse-width modulation (PWM) is a technique for controlling analog circuits with a processor's digital outputs. The average value of voltage (and current) fed to the electrical load is controlled by turning the switch between supply and load on and off at a fast pace. The longer the switch is on compared to the off periods, the higher the power supplied to the load. Proportion of on and off periods is called the duty cycle and is expressed in percent. 100% means all the time on, 0% - all the time off. Figure 3.8 shows 10%, 30%, 50% and 90% duty cycles.

There is no existing SPARK Ada compiler running on ARM system. Hence, to compile SPARK Ada program for ARM device, cross-compiler is needed. There is GNAT compiler [Hor09] created by AdaCore, but there was no cross-compiler for ARM. However, AdaCore was actively developing cross-compiler. They had working version in 2013, but tested only

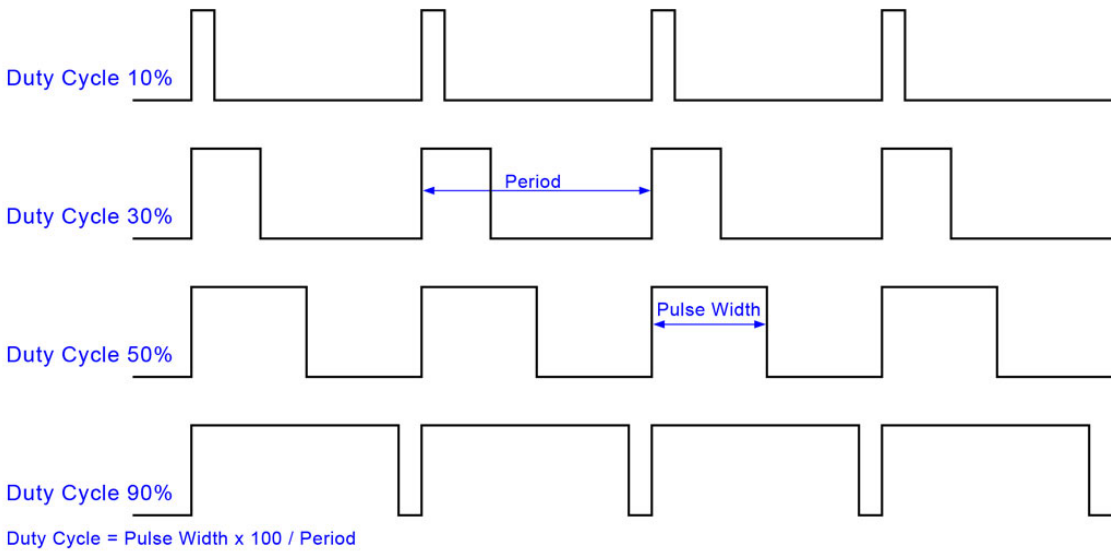

Figure 3.8: *An example of PWM duty cycles*

on their target Android-based device. In cooperation with AdaCore, cross-compiler for ARM was bundled and tested on BeagleBoard-xM. For now, GNAT cross-compiler works only on Linux 32-bit operating system.

In addition to USB ports, BeagleBoard-xM has also serial port and Ethernet port. It allows to copy programs compiled on Linux, using all three types of ports.

Chapter 4

AADL/BLESS to SPARK Ada translation

This chapter presents created AADL/BLESS to SPARK Ada translation schemes (4.1), proposed port communication (4.2) and discuss design of automatic translator, which can be created based on translation schemes (4.3). Related work in code generation from AADL, but to Java has been done in [PHR].

4.1 AADL/BLESS to SPARK Ada mapping

Mapping of AADL models to SPARK Ada is driven by "Architecture analysis & Design Language (AADL) V2 Programming Language Annex Document" [SCD14]. This document was discussed during AADL User Days in Valencia (February 2013)[1] and in Jacksonville, FL (April 2013)[2]. Ocarina tool suite (based on older AADL annex documents [HZPK08]) and its examples[3] was also helpful in understanding of AADL to Ada translation. Mapping

[1] http://www.aadl.info/aadl/downloads/committee/feb2013/presentations/13_02_04-AADL-Code%20Generation.pdf

[2] https://wiki.sei.cmu.edu/aadl/images/8/8a/Constraint_Annex_April22.v3.pdf

[3] https://github.com/yoogx/polyorb-hi-ada/tree/master/examples/aadlv2

of BLESS assertions was created in consultation with Brian Larson (BLESS creator).

4.1.1 Data types mapping

One of core AADL packages is `Base_Types`. It defines fundamental data types for AADL. Its definition is shown in listing 4.1.1.

```
package Base_Types
public

  with Data_Model;

  data Boolean
  properties
    Data_Model::Data_Representation => Boolean;
  end Boolean;

  data Integer
  properties
    Data_Model::Data_Representation => Integer;
  end Integer;

  -- Signed integer of various byte sizes

  data Integer_8 extends Integer
  properties
    Data_Model::Number_Representation => Signed;
    Source_Data_Size => 1 Bytes;
  end Integer_8;

  data Integer_16 extends Integer
  properties
    Data_Model::Number_Representation => Signed;
    Source_Data_Size => 2 Bytes;
  end Integer_16;

  data Integer_32 extends Integer
  properties
    Data_Model::Number_Representation => Signed;
    Source_Data_Size => 4 Bytes;
  end Integer_32;

  data Integer_64 extends Integer
  properties
    Data_Model::Number_Representation => Signed;
    Source_Data_Size => 8 Bytes;
  end Integer_64;

  -- Unsigned integer of various byte sizes

  data Unsigned_8 extends Integer
  properties
    Data_Model::Number_Representation => Unsigned;
    Source_Data_Size => 1 Bytes;
  end Unsigned_8;

  data Unsigned_16 extends Integer
  properties
```

```
    Data_Model::Number_Representation => Unsigned;
    Source_Data_Size => 2 Bytes;
  end Unsigned_16;

  data Unsigned_32 extends Integer
  properties
    Data_Model::Number_Representation => Unsigned;
    Source_Data_Size => 4 Bytes;
  end Unsigned_32;

  data Unsigned_64 extends Integer
  properties
    Data_Model::Number_Representation => Unsigned;
    Source_Data_Size => 8 Bytes;
  end Unsigned_64;

  data Natural extends Integer
  properties
    Data_Model::Integer_Range => 0 .. Max_Target_Integer;
  end Natural;

  data Float
  properties
    Data_Model::Data_Representation => Float;
  end Float;

  data Float_32 extends Float
  properties
    Data_Model::IEEE754_Precision => Simple;
    Source_Data_Size => 4 Bytes;
  end Float_32;

  data Float_64 extends Float
  properties
    Data_Model::IEEE754_Precision => Double;
    Source_Data_Size => 8 Bytes;
  end Float_64;

  data Character
  properties
    Data_Model::Data_Representation => Character;
  end Character;

  data String
  properties
    Data_Model::Data_Representation => String;
  end String;

end Base_Types;
```

Listing 4.1: *AADL Base_ Types package*

In Ada 2012, and thus SPARK 2014, there is package `Interfaces`, which allows for easy mapping of AADL `Base_Types` package. Mapping proposed in Annex Document [SCD14] is presented on listing 4.1.1.

```
with Interfaces;

package Base_Types is
```

```
   type AADL_Boolean is new Standard.Boolean;
   type AADL_Integer is new Standard.Integer;
   type Integer_8 is new Interfaces.Integer_8;
   type Integer_16 is new Interfaces.Integer_16;
   type Integer_32 is new Interfaces.Integer_32;
   type Integer_64 is new Interfaces.Integer_64;
   type Unsigned_8 is new Interfaces.Unsigned_8;
   type Unsigned_16 is new Interfaces.Unsigned_16;
   type Unsigned_32 is new Interfaces.Unsigned_32;
   type Unsigned_64 is new Interfaces.Unsigned_64;
   type AADL_Natural is new Standard.Integer; -- XXX incomplete range?
   type AADL_Float is new Standard.Float;
   type Float_32 is new Interfaces.IEEE_Float_32;
   type Float_64 is new Interfaces.IEEE_Float_64;
     type AADL_Character is new Standard.Character;
end Base_Types;
```

Listing 4.2: *Mapping of Base_ Types for SPARK 2014*

Target language for this thesis is SPARK 2005. SPARK 2014 tools and especially multitasking capabilities were not ready, during the time when this thesis was written. Types: `Float`, `Character` and `String` are also not part of this thesis, because of verification tools limitation. Thus, only `Integer`, `Enumeration`, `Boolean` and **`Record`** types are taken into account in mappings.

Each type is translated into simple type definition and protected type. Then it can be used in multitasking programs with Ravescar Profile. For every protected type only setter (`Put`) and getter (`Get`) subprograms are defined. It can be extended by developer during development phase. Protected objects can be also removed if they are not needed. Default value for priority, for each generated type is 10. It can be changed during development phase according to implementation details. Types: `Integer`, `Boolean` and `Natural` are already defined in SPARK Ada, thus only protected objects are generated for them. AADL `Base_Types` mapping to SPARK 2005 is presented in table 4.1.

Table 4.1: Base AADL types to SPARK mapping.

<table>
<tr><th>AADL</th><th>SPARK Ada</th></tr>
<tr><td>

```
data Integer
properties
  Data_Model::Data_Representation
     => Integer;
end Integer;
```

</td><td>

```
protected type Integer_Store
  is
      pragma Priority (10);

      function Get return Integer;
      --# global in Integer_Store;

      procedure Put(X : in Integer);
      --# global out Integer_Store;
      --# derives Integer_Store from X;
  private
      TheStoredData : Integer := 0;
  end Integer_Store;
```

</td></tr>
<tr><td>

```
data Integer_16 extends Integer
properties
  Data_Model::
     Number_Representation =>
     Signed;
  Source_Data_Size => 2 Bytes;
end Integer_16;
```

</td><td>

```
type Integer_16 is new Integer range -2**(2*8-1) .. 2**(2*8-1-1);

protected type Integer_16_Store
  is
      pragma Priority (10);

      function Get return Integer_16;
      --# global in Integer_16_Store;

      procedure Put(X : in Integer_16);
      --# global out Integer_16_Store;
      --# derives Integer_16_Store from X;
  private
      TheStoredData : Integer_16 := 0;
  end Integer_16_Store;

  protected body Integer_16_Store is
      function Get return Integer_16
      --# global in TheStoredData;
      is
      begin
          return TheStoredData;
      end Get;

      procedure Put(X : in Integer_16)
        --# global out TheStoredData;
        --# derives TheStoredData from X;
      is
      begin
          TheStoredData := X;
      end Put;
  end Integer_16_Store;
```

</td></tr>
<tr><td colspan="2" align="right">Continued on next page</td></tr>
</table>

Table 4.1 – continued from previous page

AADL	SPARK Ada

data Unsigned_16 extends Integer
properties
  Data_Model::
    Number_Representation =>
    Unsigned;
  Source_Data_Size => 2 Bytes;
end Unsigned_16;
	type Unsigned_16 is new Integer range 0 .. 2**(2*8-1);
	
	  protected type Unsigned_16_Store
	  is
	    pragma Priority (10);
	
	    function Get return Unsigned_16;
	    --# global in Unsigned_16_Store;
	
	    procedure Put(X : in Unsigned_16);
	    --# global out Unsigned_16_Store;
	    --# derives Unsigned_16_Store from X;
	  private
	    TheStoredData : Unsigned_16 := 0;
	  end Unsigned_16_Store;
	
	  protected body Unsigned_16_Store is
	    function Get return Unsigned_16
	    --# global in TheStoredData;
	    is
	    begin
	      return TheStoredData;
	    end Get;
	
	    procedure Put(X : in Unsigned_16)
	      --# global out TheStoredData;
	      --# derives TheStoredData from X;
	    is
	    begin
	      TheStoredData := X;
	    end Put;
	  end Unsigned_16_Store;

Continued on next page

Table 4.1 – continued from previous page

<table>
<tr><th>AADL</th><th>SPARK Ada</th></tr>
<tr><td>
<pre>
data Type_With_Range
  properties
    Data_Model::
     Data_Representation =>
     Integer;
    Data_Model::Base_Type => (
     classifier (Base_Types::
     Unsigned_16));
    Data_Model::Integer_Range => 0
      .. 1000;
end Type_With_Range;
</pre>
</td><td>
<pre>
type Type_With_Range is new Integer range 0 .. 1000;

  protected type Type_With_Range_Store
  is
      pragma Priority (10);

      function Get return Type_With_Range;
      --# global in Type_With_Range_Store;

      procedure Put(X : in Type_With_Range);
      --# global out Type_With_Range_Store;
      --# derives Type_With_Range_Store from X;
  private
      TheStoredData : Type_With_Range := 0;
  end Unsigned_16_Store;

  protected body Type_With_Range_Store is
      function Get return Type_With_Range
      --# global in TheStoredData;
      is
      begin
          return TheStoredData;
      end Get;

      procedure Put(X : in Type_With_Range)
        --# global out TheStoredData;
        --# derives TheStoredData from X;
      is
      begin
          TheStoredData := X;
      end Put;
  end Type_With_Range_Store;
</pre>
</td></tr>
</table>

Type range is defined using AADL properties: `Data_Model::Number_Representation`, `Source_Data_Size` and `Data_Model::Integer_Range`. When `Data_Model::Integer_Range` property is not specified, then range is calculated. In case of `Integer` representation range starts from negative value, for `Unsigned` - from 0. Maximum value for `Integer` is calculated using the formula 4.1.

$$\text{Integer\_[Number\_Of\_Bytes * 8]\_Max} = 2^{\text{Number\_Of\_Bytes}*8-1} - 1 \tag{4.1}$$

The minimum value formula for Integer (4.2) and maximum value for `Unsigned` (4.3) use similar strategy.

$$\text{Integer\_[Number\_Of\_Bytes * 8]\_Min} = -2^{\text{Number\_Of\_Bytes}*8-1} \tag{4.2}$$

$$\text{Unsigned\_[Number\_Of\_Bytes * 8]\_Max} = 2^{\text{Number\_Of\_Bytes}*8} - 1 \tag{4.3}$$

Mapping for enumeration types, presented on table 4.2, is pretty straightforward. BLESS properties are ignored in translation. In addition to simple types, protected types are generated.

Continued on next page

Table 4.2 – continued from previous page

AADL	SPARK Ada

Table 4.2: AADL/BLESS enumeration types to SPARK mapping.

AADL	SPARK Ada

`data Enum_Type`
`  properties`
`    BLESS::Typed=>"enumeration (Enumerator1,`
`     Enumerator2, Enumerator3)";`
`    Data_Model::Data_Representation => Enum;`
`    Data_Model::Enumerators => ("Enumerator1", "`
`     Enumerator2", "Enumerator3");`
`end Enum_Type;`
	`type Enum_Type is (Enumerator1, Enumerator2,`
	`     Enumerator3);`
	
	`protected type Enum_Type_Store`
	`  is`
	`      pragma Priority (10);`
	
	`      function Get return Enum_Type;`
	`      --# global in Enum_Type_Store;`
	
	`      procedure Put(X : in Enum_Type);`
	`      --# global out Enum_Type_Store;`
	`      --# derives Enum_Type_Store from X;`
	`  private`
	`      TheStoredData : Enum_Type := Enum_Type'First;`
	`  end Enum_Type_Store;`
	
	`  protected body Enum_Type_Store is`
	`      function Get return Enum_Type`
	`      --# global in TheStoredData;`
	`      is`
	`      begin`
	`          return TheStoredData;`
	`      end Get;`
	
	`      procedure Put(X : in Enum_Type)`
	`        --# global out TheStoredData;`
	`        --# derives TheStoredData from X;`
	`      is`
	`      begin`
	`          TheStoredData := X;`
	`      end Put;`
	`  end Enum_Type_Store;`

Sometimes it is pragmatic to define a type, which has exactly the same range like some already existing type. Especially when it is used for some specific calculations. E.g. measuring the speed. Let's say, that `Unsigned_16` was used. Then, during development of next car model, it becomes not enough. In case when e.g. `Speed_Type` is not defined, there are two possible resolutions. First: change definition (range) of `Unsigned_16`. That is bad choice, especially because its name specify the range. Another reason: it might be used not only for measuring the Speed, but maybe also for fuel level, which range is still fine. Second option is to change `Unsigned_16` to e.g. `Unsigned_32` everywhere in Speed Control Module (and maybe also in some external modules). When `Speed_Type` is defined and used everywhere for speed units, then only definition of `Speed_Type` has to be changed. To define type, using existing type in AADL, derived type (defined with `extends` keyword) or `Data_Model::Base_Type` property can be used. Translation to SPARK Ada is shown in table 4.3. There are two ways to define type based on some other type in SPARK Ada:

- subtype - it is compatible with its parent, in other words: parent type variable can be assigned to it, if its value is in the subtype range
- derived type - it is incompatible with its parent (parent type variable cannot be assigned to it), but inherits its primitive operations

Table 4.3: AADL types to SPARK mapping: Subtypes.

<table>
<tr><th>AADL</th><th>SPARK Ada</th></tr>
<tr><td><pre>
data Speed_Type extends Base_Types::Integer
end Speed_Type;
</pre></td><td><pre>
subtype Speed_Type is Base_Types.Integer;
</pre></td></tr>
<tr><td><pre>
data Speed_Type
  properties
    BLESS::Typed=>"integer";
      Data_Model::Base_Type => (classifier(
     Base_Types::Unsigned_16));
end Speed_Type;
</pre></td><td><pre>
type Speed_Type is new Base_Types.Unsigned_16;
</pre></td></tr>
</table>

AADL array type can be defined using property `Data_Model::Data_Representation`. In addition to that, size for array has to be specified by `Data_Model::Dimension` property. Sample mapping of array of 10 integers is shown in table 4.4.

Table 4.4: AADL arrays to SPARK Ada mapping

<table>
<tr><th>AADL</th><th>SPARK Ada</th></tr>
<tr><td>

```
data Some_Array
    properties
      BLESS::Typed => "array [10] of Base_Types::
     Integer_32";
      Data_Model::Data_Representation => Array;
      Data_Model::Base_Type => (classifier(
     Base_Types::Integer_32));
      Data_Model::Dimension => (10);
end Some_Array;
```

</td><td>

```
subtype Some_Array_Index is Integer range 1 ..
     10;
  type Some_Array is array (Some_Array_Index) of
     Base_Types.Integer_32;

  protected type Some_Array_Store
  is
      pragma Priority (10);

      function Get(Ind : in Integer) return
     Base_Types.Integer_32;
      --# global in Some_Array_Store;

      procedure Put(Ind : in Integer; Val : in
     Base_Types.Integer_32);
      --# global in out Some_Array_Store;
      --# derives Some_Array_Store from
     Some_Array_Store, Ind, Val;
  private
      TheStoredData : Some_Array := Some_Array'(
     others => 0);
  end Some_Array_Store;

  protected body Some_Array_Store
  is
      function Get(Ind : in Integer) return
     Base_Types.Integer_32
      --# global in TheStoredData;
      is
      begin
          return TheStoredData(Ind);
      end Get;

      procedure Put(Ind : in Integer; Val : in
     Base_Types.Integer_32)
        --# global in out TheStoredData;
        --# derives TheStoredData from
     TheStoredData, Ind, Val;
      is
      begin
          TheStoredData(Ind) := Val;
      end Put;
  end Some_Array_Store;
```

</td></tr>
</table>

AADL v2 allows to create struct data types, using `Data_Model::Data_Representation => Struct`. AADL Struct is mapped to SPARK Ada record type. The mapping is presented in table

4.5.

Table 4.5: AADL struct to SPARK Ada record mapping

<table>
<tr><th>AADL</th><th>SPARK Ada</th></tr>
<tr><td><pre>
data Some_Record_Type
  properties
    BLESS::Typed => "record (
      Field1 : Base_Types::Integer_32;
      Field2 : Base_Types::Boolean;
      Field3 : Base_Types::Unsigned_32;
      );
    Data_Model::Data_Representation => Struct;
    Data_Model::Element_Names => ("Field1", "Field2
     ", "Field3");
    Data_Model::Base_Type =>
    (
      classifier(Base_Types::Integer_32),
      classifier(Base_Types::Boolean),
      classifier(Base_Types::Unsigned_32)
    );
end Some_Record_Type;
</pre></td><td><pre>
type Some_Record_Type is record
      Field1 : Integer_32;
      Field2 : Boolean;
      Field3 : Unsigned_32;
  end record;
</pre></td></tr>
</table>

Data types translations are created based on Brian Larson's AADL/BLESS models of PCA Pump. They are syntacticly verified with SPARK Examiner. During development of types mapping, SPARK Examiner was helpful also for detecting inconsistencies in AADL models. Eg. it detected redundancy in enumerators. Both `Alarm_Type` and `Warning_Type` contained `No_Alarm` enumerator, which was a bug. All enumerators, for all types have to be unique. Thus `Warning_Type` should have `No_Warning` enumerator instead.

4.1.2 AADL ports mapping

Proposed ports mapping shown in table 4.6 is based on AADL runtime services from Annex 2 to "Programming Language Annex Document" [SCD14]. Additionally, the mapping contains SPARK 2005 contracts. Data types used by ports has to be defined earlier, to be visible. Moreover, for port communication, protected types are used, to enable concurrency.

Table 4.6: AADL to SPARK ports mapping.

AADL/BLESS	SPARK Ada

Port_Name :
  in data port Port_Type;
	-- spec (.ads):
	--# own protected Port_Name : Port_Type_Store(Priority => 10)
	
	procedure Receive_Port_Name;
	--# global out Port_Name;
	
	-- body (.adb):
	Port_Name : Port_Type_Store;
	
	procedure Receive_Port_Name
	is
	begin
	  -- TODO: implement receiving Port_Name value
	  -- e.g.:
	  -- Port_Name.Put(Some_Pkg.Get_Port_Name);
	end Receive_Port_Name;

Port_Name :
  out data port Port_Type;
	-- spec (.ads)
	--# own protected Port_Name : Port_Type_Store(Priority => 10)
	
	procedure Get_Port_Name(Port_Name_Out : **out** Port_Type);
	--# global in Port_Name;
	--# derives Port_Name_Out from Port_Name;
	
	-- body (.adb):
	Port_Name : Port_Type_Store;
	
	procedure Get_Port_Name(Port_Name_Out : **out** Port_Type)
	is
	begin
	  Port_Name_Out := Port_Name.Get;
	end Get_Port_Name;

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

Continued on next page

Table 4.6 – continued from previous page

<table>
<tr><th>AADL/BLESS</th><th>SPARK Ada</th></tr>
<tr><td><pre>Port_Name :
  out event port;</pre></td><td><pre>-- 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;
end Send_Port_Name;</pre></td></tr>
<tr><td><pre>Port_Name :
  in event data port Port_Type;</pre></td><td><pre>-- spec (.ads)
--# own protected Port_Name : Port_Type_Store(Priority => 10);

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_Store;

procedure Put_Port_Name (Port_Name_In : Port_Type)
is
begin
  Port_Name.Put(Port_Name_In);
end Put_Port_Name;</pre></td></tr>
<tr><td><pre>Port_Name :
  out event data port Port_Type;</pre></td><td><pre>-- spec (.ads)
--# own protected Port_Name : Port_Type_Store(Priority => 10);

procedure Send_Port_Name;
  --# global in Port_Name;

-- body (.adb):
Port_Name : Port_Type_Store;

procedure Send_Port_Name
is
begin
  -- TODO: implement receiving Port_Name value
  -- e.g.:
  -- Some_Pkg.Put_Port_Name(Port_Name);
end Send_Port_Name;</pre></td></tr>
</table>

4.1.3 Thread to task mapping

AADL Threads are mapped into SPARK Ada tasks according to table 4.7. Communication between threads is described in section 4.2.1.

Table 4.7: *AADL threads to SPARK Ada tasks mapping.*

<table>
<tr><th>AADL/BLESS</th><th>SPARK Ada</th></tr>
<tr><td><pre>
package Some_Pkg
  thread Some_Thread
    features
      Some_Port : out data port Port_Type;
  end Some_Thread;

  thread implementation Some_Thread.imp
  end Some_Thread.imp;
end Some_Pkg;
</pre></td><td><pre>
package Some_Pkg
is
  task type Some_Thread
  --# global out Some_Port;
  is
    pragma Priority(10);
  end Some_Thread;
end Some_Pkg;

package body Some_Pkg
is
  st : Some_Thread;

  task body Some_Thread
  is
  begin
    loop
      -- implementation
    end loop;
  end Some_Thread;
end Some_Pkg;
</pre></td></tr>
</table>

4.1.4 Subprograms mapping

Mappings of subprograms is also straightforward. However, it is different than mapping proposed in "AADL Code Generation Annex" [SCD14]. It does not use **renames** clause, but mapping directly to subprogram specification and body. For now, body is empty, because behavior (implementation) is not captured. Subprogram mapping should be revised and consulted with AADL committee members, in order to understand their design decisions.

Table 4.8: *AADL subprograms to SPARK Ada subprograms 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);`
	
	`procedure sp(e : in T; s : out T) is`
	`begin`
	`  --# implementation`
	`end sp;`

4.1.5 Feature groups mapping

In SPARK Ada there are nested packages and child packages. Sample nested packages are shown in listing 4.1.5. Equivalent child packages are shown in listing 4.1.5. The name of a 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 4.1.5 and 4.1.5). 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 4.1.5 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 4.3: *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 4.4: *Child packages in SPARK Ada*

There was an idea to create child package to encapsulate one feature group in it. However, SPARK Ada does not allow to access child packages private part from parent. Thus, it will require to expose feature group internal variable as public. It is definitely not good solution. Thus, feature group is translated with prefix `Feature_Group_Name_*`. Feature group mapping is presented in section 4.1.6, in listings 4.1.6 and 4.1.6.

4.1.6 AADL package to SPARK Ada package mapping

On listing 4.1.6, there is sample AADL package with system. It contains all types of ports described in section 4.1.2 and one feature group with two ports as example of feature group mapping.

```
package Some_Pkg
public
with Base_Types;

feature group Some_Features
features
  Some_Out_Port: out data port Base_Types::Integer;
```

```
  Some_In_Port: in data port Base_Types::Integer;
end Some_Features;

system Some_System
features
  Some_Feature_Group : feature group Some_Features;

  In_Data_Port : in data port Base_Types::Integer;
  Out_Data_Port : out data port Base_Types::Integer;
  In_Event_Port : in event port;
  Out_Event_Port : out event port;
  In_Event_Data_Port : in event data port Base_Types::Integer;
  Out_Event_Data_Port : out event data port Base_Types::Integer;
end Some_System;

end Some_Pkg;
```

Listing 4.5: *Sample AADL package with system*

For now, only single process SPARK Ada application is considered. Thus, ports are exposed only on system level. Communication between threads in process will be realized by protected objects and only SPARK annotations and data types will be needed as described in section 4.1.3. Based on ports mapping, presented in section 4.1.2, translation to SPARK Ada package is shown in listing 4.1.6.

```
package Some_Pkg
--# own Some_Features_Some_Out_Port : Integer;
--#     Some_Features_Some_In_Port : Integer;
--#     In_Data_Port : Integer;
--#     Out_Data_Port : Integer;
--#     In_Event_Data_Port : Integer;
--#     Out_Event_Data_Port : Integer;
--# initializes Some_Features_Some_Out_Port,
--#             Some_Features_Some_In_Port,
--#             In_Data_Port,
--#             Out_Data_Port,
--#             In_Event_Data_Port,
--#             Out_Event_Data_Port;
is

    function Some_Features_Get_Some_Out_Port return Integer;
    --# global in Some_Features_Some_Out_Port;

    procedure Some_Features_Receive_Some_In_Port;
    --# global out Some_Features_Some_In_Port;

    procedure Receive_In_Data_Port;
    --# global out In_Data_Port;

    function Get_Out_Data_Port return Integer;
    --# global in Out_Data_Port;

    procedure Put_In_Event_Port;

    procedure Send_Out_Event_Port;
```

```
    procedure Put_In_Event_Data_Port(In_Event_Data_Port_In : Integer);
    --# global out In_Event_Data_Port;
    --# derives In_Event_Data_Port from In_Event_Data_Port_In;

    procedure Send_Out_Event_Data_Port;
    --# global in Out_Event_Data_Port;

end Some_Pkg;

package body Some_Pkg
is
    Some_Features_Some_Out_Port : Integer := 0;
    Some_Features_Some_In_Port : Integer := 0;
    In_Data_Port : Integer := 0;
    Out_Data_Port : Integer := 0;
    In_Event_Data_Port : Integer := 0;
    Out_Event_Data_Port : Integer := 0;

    function Some_Features_Get_Some_Out_Port return Integer
    is
    begin
        return Some_Features_Some_Out_Port;
    end Some_Features_Get_Some_Out_Port;

    procedure Some_Features_Receive_Some_In_Port
    is
    begin
        -- implementation
    end Some_Features_Receive_Some_In_Port;

    procedure Receive_In_Data_Port
    is
    begin
        -- implementation
    end Receive_In_Data_Port;

    function Get_Out_Data_Port return Integer
    is
    begin
        return Out_Data_Port;
    end Get_Out_Data_Port;

    procedure Put_In_Event_Port
    is
    begin
        -- implementation
    end Put_In_Event_Port;

    procedure Send_Out_Event_Port
    is
    begin
        -- implementation
    end Send_Out_Event_Port;

    procedure Put_In_Event_Data_Port(In_Event_Data_Port_In : Integer)
    is
    begin
        In_Event_Data_Port := In_Event_Data_Port_In;
    end Put_In_Event_Data_Port;

    procedure Send_Out_Event_Data_Port
    is
    begin
        -- implementation
    end Send_Out_Event_Data_Port;
```

```
end Some_Pkg;
```

Listing 4.6: *Translation of sample AADL package from listing 4.1.6*

4.1.7 AADL property set to SPARK Ada package mapping

There is no equivalent construct for AADL property set in SPARK Ada. Thus property set is mapped to SPARK Ada package. In this thesis, only properties of type `constant aadlinteger` are considered. There are issues with using non-constant types in SPARK Ada package (e.g. when using them in some type definition). Table 4.9 shows sample property set mapping to SPARK Ada package.

Table 4.9: *AADL property set to SPARK Ada package mapping*

AADL	SPARK Ada

`property set Some_Properties is`
`  Some_Property1 : constant aadlinteger => 10;`
`  Some_Property2 : constant aadlinteger => 27`
`    applies to (all);`
`  Some_Property3 : constant aadlinteger =>`
`    Some_Properties::Some_Property1 applies to (`
`    all);`
`end Some_Properties;`
	`package Some_Properties`
	`is`
	`  Some_Property1 : constant Integer := 10;`
	`  Some_Property2 : constant Integer := 27;`
	`  Some_Property3 : constant Integer :=`
	`    Some_Property1;`
	`end Some_Properties;`

In AADL, all declarations must have an `applies to` clause. It is ignored in resulted SPARK Ada translation. However, it can be used in the future e.g. for automatic generation of `with` clauses.

4.1.8 BLESS mapping

In cooperation with Brian Larson, translations for BLESS assertions, invariant, pre- and postconditions were created. Table 4.10 presents their mapping to SPARK Ada. Generated (translated) code may not be complete. Then developer's effort to implement missing parts will be required. E.g. when assertion is specified in AADL/BLESS model, but not defined, it has to be implemented.

Table 4.10: BLESS to SPARK contracts mapping

<table>
<tr><th>AADL/BLESS</th><th>SPARK Ada</th></tr>
<tr><td><pre>BLESS::Assertion=>"<<COND1()>>"</pre></td><td><pre>--# assert COND1;</pre></td></tr>
<tr><td><pre>thread Some_Thread
features
  Some_Port : out event port
  {BLESS:Assertion => "<<(Var1 < Var2 and COND2())
     >>";};
end Some_Thread;</pre></td><td><pre>task body Some_Thread
is
begin
  loop
    --# assert (Var1 < Var2 and COND2);
  end loop;
end Some_Thread;</pre></td></tr>
<tr><td><pre>thread implementation Some_Thread.imp
annex BLESS
{**
  invariant <<(Some_Var < Other_Var)>>
**};
end Some_Thread.imp;</pre></td><td><pre>task body Some_Thread
is
begin
  loop
    --# assert (Some_Var < Other_Var);
  end loop;
end Some_Thread;</pre></td></tr>
<tr><td><pre>thread implementation Some_Thread.imp
annex BLESS
{**
  assert
  <<State1 : :COND1() or COND2()>>
  <<Var : :=
            (State1()) -> 0,
            (State2()) -> -1,
            (State3()) -> 9
  >>
**};
end Some_Thread.imp;</pre></td><td><pre>task body Some_Thread
is
begin
  loop
    --# assert (COND1 or COND2)
    --#   -> State1();
    --# assert (Var = 0) -> State1 and
    --#   (Var = -1) -> State2 and
    --#   (Var = 9) -> State3;
  end loop;
end Some_Thread;</pre></td></tr>
</table>

Continued on next page

Table 4.10 – continued from previous page

AADL/BLESS	SPARK Ada

<pre>subprogram Some_Subprogram
features
  param : out parameter Base_Types::Integer;
annex subBless
{**
  pre <<(param > 0)>>
  post <<(param = 0)>>
**};
end Some_Subprogram;</pre>
	<pre>procedure Some_Subprogram(Param : in out Integer)
	    ;
	  --# pre Param > 0;
	  --# post Param = 0;</pre>

4.2 Port-based communication

Communication between AADL components is realized by ports. AADL ports can be declared in subprograms, threads, processes, systems and other entities. In this section communication between threads in single-process SPARK Ada application (4.2.1) and concept of communication between two systems (4.2.2) are presented.

4.2.1 Threads communication

Example of communication between threads, in single process is depicted in the figure 4.2.1. There are two threads (`some_thread` and `other_thread`) in one process. AADL model and its translation to SPARK Ada is presented in the table 4.11. Connection between threads has to be specified in process implementation. Based on mappings from section 4.1, protected object is defined, but subprograms are skipped, because communication takes place only internally. The result of translation consists of two tasks and private global protected object, which enable communication between them. Additionally, both tasks has global annotation (one with **out** mode, other with **in** mode), which announce use of protected object in their bodies.

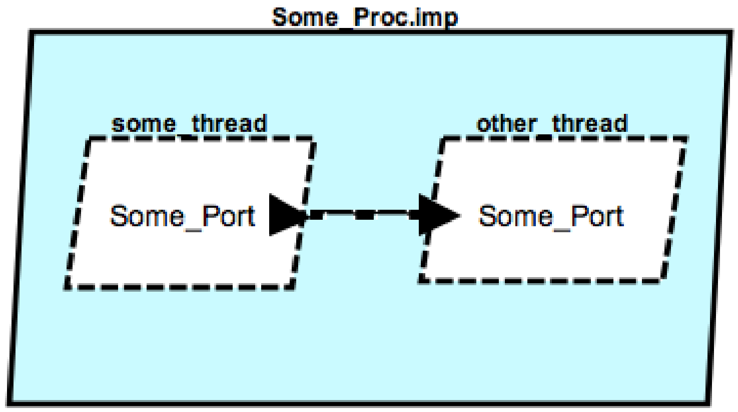

Figure 4.1: *Example of port communication between threads*

Threads can be also placed in different packages. The same example of two threads within one process, but in different packages is presented in table 4.12. In this case, subprograms present in mapping table, in section 4.2 are also present in resulted translation. Moreover, body of procedure `Receive_Some_Port` is implemented as a result of defined connection between threads in the process implementation, in AADL model.

Table 4.11: *AADL threads communication to SPARK Ada tasks communication translation*

AADL/BLESS

```
package Some_Pkg
public
with Base_Types;

  process Some_Proc
  end Some_Proc;

  process implementation Some_Proc.imp
  subcomponents
    some_thread: thread Some_Thread.imp;
    other_thread: thread Other_Thread.imp;
  connections
    connection: port some_thread.Some_Port ->
     other_thread.Some_Port;
  end Some_Proc.imp;

  thread Some_Thread
    features
      Some_Port : out data port Base_Types::
     Integer;
  end Some_Thread;

  thread implementation Some_Thread.imp
  end Some_Thread.imp;

  thread Other_Thread
    features
      Some_Port : in data port Base_Types::Integer
     ;
  end Other_Thread;

  thread implementation Other_Thread.imp
  end Other_Thread.imp;

end Some_Pkg;
```

SPARK Ada

```
with Base_Types;
--# inherit Base_Types;
package Some_Pkg
--# own task st : Some_Thread;
--#     task ot : Other_Thread;
--#     protected Some_Port : Base_Types.
     Integer_Store (Priority => 10);
is

private
   task type Some_Thread
   --# global out Some_Port;
   is
      pragma Priority (10);
   end Some_Thread;

   task type Other_Thread
   --# global in Some_Port;
   is
      pragma Priority (10);
   end Other_Thread;
end Some_Pkg;

package body Some_Pkg
is
   st : Some_Thread;
   ot : Other_Thread;
   Some_Port : Base_Types.Integer_Store;

   task body Some_Thread
   is
   begin
      loop
         -- implementation
      end loop;
   end Some_Thread;

   task body Other_Thread
   is
   begin
      loop
         -- implementation
      end loop;
   end Other_Thread;
end Some_Pkg;
```

Table 4.12: AADL threads communication to SPARK Ada tasks communication translation (multiple packages)

<table>
<tr><th>AADL/BLESS</th><th>SPARK Ada</th></tr>
<tr><td>
<pre>
package Pkg1
public
with Base_Types, Pkg2;

  process Some_Proc
  end Some_Proc;

  process implementation
     Some_Proc.imp
  subcomponents
    some_thread: thread
     Some_Thread.imp;
    other_thread: thread Pkg2::
     Other_Thread.imp;
  connections
    connection: port some_thread.
     Some_Port -> other_thread.
     Some_Port;
  end Some_Proc.imp;

  thread Some_Thread
    features
      Some_Port : out data port
     Base_Types::Integer;
  end Some_Thread;

  thread implementation
     Some_Thread.imp
  end Some_Thread.imp;
</pre>
</td><td>
<pre>
with Base_Types;
--# inherit Base_Types;
package Pkg1
--# own task st : Some_Thread;
--#   protected Some_Port : Base_Types.Integer_Store (Priority =>
     10);
is
   procedure Get_Some_Port(Some_Port_Out : out Integer);
   --# global in Some_Port;
   --# derives Some_Port_Out from Some_Port;

private
   task type Some_Thread
   --# global out Some_Port;
   is
      pragma Priority (10);
   end Some_Thread;
end Pkg1;

package body Pkg1
is
   st : Some_Thread;
   Some_Port : Base_Types.Integer_Store;

   procedure Get_Some_Port(Some_Port_Out : out Integer)
   is
   begin
      Some_Port_Out := Some_Port.Get;
   end Get_Some_Port;

   task body Some_Thread
   is
   begin
      loop
         -- implementation
      end loop;
   end Some_Thread;

end Pkg1;
</pre>
</td></tr>
<tr><td colspan="2" align="right">Continued on next page</td></tr>
</table>

Table 4.12 – continued from previous page

<table>
<tr><th>AADL/BLESS</th><th>SPARK Ada</th></tr>
<tr><td>
<pre>
package Pkg2
public
with Base_Types;

  thread Other_Thread
    features
      Some_Port : in data port
     Base_Types::Integer;
  end Other_Thread;

  thread implementation
     Other_Thread.imp
  end Other_Thread.imp;
end Pkg2;
</pre>
</td><td>
<pre>
with Base_Types;
with Pkg1;
--# inherit Base_Types,
--#   Pkg1;
package Pkg2
--# own task ot : Other_Thread;
--#   protected Some_Port : Base_Types.Integer_Store (Priority =>
     10);
is
   procedure Receive_Some_Port;
   --# global out Some_Port;
   --#   in Pkg1.Some_Port;

private
   task type Other_Thread
   --# global in Some_Port;
   is
      pragma Priority (10);
   end Other_Thread;
end Pkg2;

package body Pkg2
is
   ot : Other_Thread;
   Some_Port : Base_Types.Integer_Store;

   procedure Receive_Some_Port
   is
      Temp : Integer;
   begin
      Pkg1.Get_Some_Port(Temp);
      Some_Port.Put(Temp);
   end Receive_Some_Port;

   task body Other_Thread
   is
   begin
      loop
         -- implementation
      end loop;
   end Other_Thread;
end Pkg2;
</pre>
</td></tr>
</table>

In the given example, communication is one way: from `Pkg1` package to `Pkg2` package. Thus, `Pkg1` package does not need to know that `Pkg2` package exists. In other words: it does not need to "with" it. However, if two way communication is needed (between `Pkg1` to `Pkg2`), then `Pkg1` package has to "with" `Pkg2` package. It is not a case in first example, where communication between threads take place in the same package. Modified model of second example, with communication from `Pkg2` to `Pkg1`, is depicted in the figure 4.2.1 and presented in listing 4.2.1.

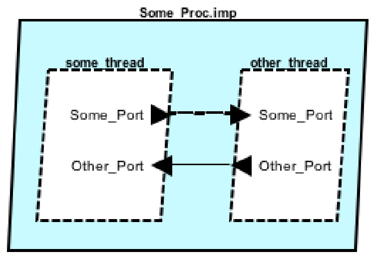

Figure 4.2: *Example of two way port communication between threads in different packages*

```
package Pkg1TwoWay
public
with Base_Types, Pkg2TwoWay;

  process Some_Proc
  end Some_Proc;

  process implementation Some_Proc.imp
  subcomponents
    some_thread: thread Some_Thread.imp;
    other_thread: thread Pkg2TwoWay::Other_Thread.imp;
  connections
    connection: port some_thread.Some_Port -> other_thread.Some_Port;
    connection2: port some_thread.Other_Port -> other_thread.Other_Port;
  end Some_Proc.imp;

  thread Some_Thread
    features
      Some_Port : out data port Base_Types::Integer;
      Other_Port : in data port Base_Types::Integer;
  end Some_Thread;

  thread implementation Some_Thread.imp
  end Some_Thread.imp;
end Pkg1TwoWay;

package Pkg2TwoWay
public
with Base_Types;
  thread Other_Thread
    features
      Some_Port : in data port Base_Types::Integer;
      Other_Port : out data port Base_Types::Integer;
  end Other_Thread;

  thread implementation Other_Thread.imp
  end Other_Thread.imp;
end Pkg2TwoWay;
```

Listing 4.7: *AADL model of two way port communication threads in different packages*

This model, translated to SPARK Ada is presented in listing 4.2.1. It will not compile. GNAT compiler returns `circular unit dependency` error. Additionally verification with SPARK Examiner returns error: `Semantic Error 135 - The package Pkg2TwoWay is undeclared or not visible, or there is a circularity in the list of inherited packages.` Now, the problem is that two-way communication is allowed in AADL, but not in SPARK, nor even in Ada. This require further investigation, which is omitted in this thesis.

```
with Base_Types;
with Pkg2TwoWay;
--# inherit Base_Types,
--#         Pkg2TwoWay;
package Pkg1TwoWay
--# own task st : Some_Thread;
--#     protected Some_Port : Base_Types.Integer_Store (Priority => 10);
--#     protected Other_Port : Base_Types.Integer_Store (Priority => 10);
is
   procedure Get_Some_Port(Some_Port_Out : out Integer);
   --# global in Some_Port;
   --# derives Some_Port_Out from Some_Port;

   procedure Receive_Other_Port;
   --# global out Other_Port;
   --#        in Pkg2TwoWay.Other_Port;

private
   task type Some_Thread
   --# global out Some_Port;
   is
      pragma Priority (10);
   end Some_Thread;
end Pkg1TwoWay;

package body Pkg1TwoWay
is
   st : Some_Thread;
   Some_Port : Base_Types.Integer_Store;
   Other_Port : Base_Types.Integer_Store;

   procedure Get_Some_Port(Some_Port_Out : out Integer)
   is
   begin
      Some_Port_Out := Some_Port.Get;
   end Get_Some_Port;

   procedure Receive_Other_Port
   is
      Temp : Integer;
   begin
      Pkg2TwoWay.Get_Other_Port(Temp);
      Other_Port.Put(Temp);
   end Receive_Other_Port;

   task body Some_Thread
   is
   begin
      loop
```

```
         -- implementation
         null;
      end loop;
   end Some_Thread;

end Pkg1TwoWay;

with Base_Types;
with Pkg1TwoWay;
--# inherit Base_Types,
--#         Pkg1TwoWay;
package Pkg2TwoWay
--# own task ot : Other_Thread;
--#     protected Some_Port : Base_Types.Integer_Store (Priority => 10);
--#     protected Other_Port : Base_Types.Integer_Store (Priority => 10);
is
   procedure Receive_Some_Port;
   --# global out Some_Port;
   --#        in Pkg1TwoWay.Some_Port;

   procedure Get_Other_Port(Other_Port_Out : out Integer);
   --# global in Other_Port;
   --# derives Other_Port_Out from Other_Port;

private
   task type Other_Thread
   --# global in Some_Port;
   is
      pragma Priority (10);
   end Other_Thread;
end Pkg2TwoWay;

package body Pkg2TwoWay
is
   ot : Other_Thread;
   Some_Port : Base_Types.Integer_Store;
   Other_Port : Base_Types.Integer_Store;

   procedure Receive_Some_Port
   is
      Temp : Integer;
   begin
      Pkg1TwoWay.Get_Some_Port(Temp);
      Some_Port.Put(Temp);
   end Receive_Some_Port;

   procedure Get_Other_Port(Other_Port_Out : out Integer)
   is
   begin
      Other_Port_Out := Other_Port.Get;
   end Get_Other_Port;

   task body Other_Thread
   is
   begin
      loop
         -- implementation
         null;
      end loop;
   end Other_Thread;

end Pkg2TwoWay;
```

Listing 4.8: *Two way port communication translated to SPARK Ada*

4.2.2 Systems communication

This section is a concept, how communication between different systems can look like. AADL system consists of process(es) and process consists of threads. Ports would be exposed by package if they are specified in system entity. Communication between two systems cab be described by another system. Figure 4.2.2 presents communication between two systems: panel and pump. AADL model of this system comprises 3 packages: `Main`, `Panel` and `Pump`. They are presented in listing 4.2.2. `Panel` package has one thread `Panel_Thread` with two **out** ports: `event port` and `event data port`. Both ports are exposed by process `panel_process` and then by system `panel`. `Pump` package has similar structure, but two **in** ports. Both are also exposed by process (`pump_process`) and system (`pump`). Connection between these two packages are defined in `Main` package.

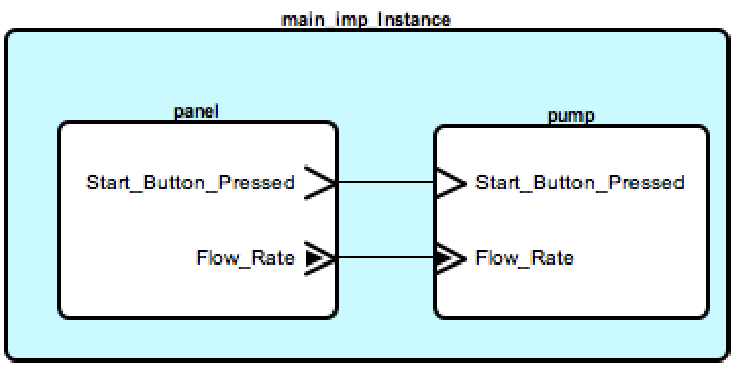

Figure 4.3: *Example of port communication between systems*

```
package Panel
public
with Base_Types;

  thread Panel_Thread
  features
    Start_Button_Pressed: out event port;
    Flow_Rate: out event data port Base_Types::Integer;
  end Panel_Thread;

  thread implementation Panel_Thread.imp
  end Panel_Thread.imp;

  process panel_process
```

```
    features
      Start_Button_Pressed: out event port;
      Flow_Rate: out event data port Base_Types::Integer;
  end panel_process;

  process implementation panel_process.imp
    subcomponents
      panel_thread: thread Panel_Thread.imp;
    connections
      sbp: port panel_thread.Start_Button_Pressed->Start_Button_Pressed;
      fr: port panel_thread.Flow_Rate->Flow_Rate;
  end panel_process.imp;

  system panel
    features
      Start_Button_Pressed: out event port;
      Flow_Rate: out event data port Base_Types::Integer;
  end panel;

  system implementation panel.imp
    subcomponents
      panel_process: process panel_process.imp;
    connections
      sbp: port panel_process.Start_Button_Pressed->Start_Button_Pressed;
      fr: port panel_process.Flow_Rate->Flow_Rate;
  end panel.imp;

end Panel;

package Pump
public
with Base_Types;

  thread Rate_Controller
    features
      Start_Button_Pressed: in event port;
      Flow_Rate: in event data port Base_Types::Integer;
  end Rate_Controller;

  thread implementation Rate_Controller.imp
  end Rate_Controller.imp;

  process pump_process
    features
      Start_Button_Pressed : in event port;
      Flow_Rate: in event data port Base_Types::Integer;
  end pump_process;

  process implementation pump_process.imp
    subcomponents
      Rate_Controller: thread Rate_Controller.imp;
    connections
      sbp: port Start_Button_Pressed->Rate_Controller.Start_Button_Pressed;
      fr: port Flow_Rate->Rate_Controller.Flow_Rate;
  end pump_process.imp;

  system pump
    features
      Start_Button_Pressed : in event port;
      Flow_Rate: in event data port Base_Types::Integer;
  end pump;

  system implementation pump.imp
    subcomponents
      pump_process : process pump_process.imp;
```

```
    connections
      sbp: port Start_Button_Pressed->pump_process.Start_Button_Pressed;
      fr: port Flow_Rate->pump_process.Flow_Rate;
  end pump.imp;
end Pump;

package Main
public
with Pump,
  Panel;

  system main
  end main;

  system implementation main.imp
    subcomponents
      panel: system Panel::panel.imp;
      pump: system Pump::pump.imp;
    connections
      sbp2sbp: port panel.Start_Button_Pressed->pump.Start_Button_Pressed;
      fr2fr: port panel.Flow_Rate->pump.Flow_Rate;
  end main.imp;
end Main;
```

Listing 4.9: *AADL model of port communication between systems*

Based on mappings from section 4.1, conforming SPARK Ada code is presented in listing 4.2.2. There are two packages: `Panel` and `Pump`. `Main` package is omitted. Both contain procedures representing ports interfaces, according to ports mapping from section 4.1.2. Additionally, both consists of empty thread declarations and bodies, which conforms to translations from section 4.1.3. However, in this case, both packages will work in different systems, thus different processes. To enable communication between different systems, deployment methodology and the middle-ware layer has to be created. It will allow not only for system to system communication, but also for communication with devices. It is omitted in this thesis.

```
with Pump;
with Base_Types;
--# inherit Pump,
--#         Base_Types;
package Panel
--# own task pt : Panel_Thread;
--#     protected Flow_Rate : Base_Types.Integer_Store (Priority => 10);
is
    procedure Send_Start_Button_Pressed;

    procedure Send_Flow_Rate;
    --# global in Flow_Rate;
    --#        out Pump.Flow_Rate;
```

```
private
    task type Panel_Thread
    --# global in out Flow_Rate;
    is
        pragma Priority (10);
    end Panel_Thread;

end Panel;

package body Panel
is
    pt : Panel_Thread;
    Flow_Rate : Base_Types.Integer_Store;

    procedure Send_Start_Button_Pressed
    is
    begin
        Pump.Put_Start_Button_Pressed;
    end Send_Start_Button_Pressed;

    procedure Send_Flow_Rate
    is
        Flow_Rate_Temp : Integer;
    begin
        Flow_Rate_Temp := Flow_Rate.Get;
        Pump.Put_Flow_Rate(Flow_Rate_Temp);
    end Send_Flow_Rate;

    task body Panel_Thread
    is
    begin
      loop
         -- implementation
      end loop;
    end Panel_Thread;
end Panel;

with Base_Types;
--# inherit Base_Types;
package Pump
--# own task rc : Rate_Controller;
--#     protected Flow_Rate : Base_Types.Integer_Store (Priority => 10);
is
    procedure Put_Start_Button_Pressed;

    procedure Put_Flow_Rate(Flow_Rate_In : Integer);
    --# global out Flow_Rate;
    --# derives Flow_Rate from Flow_Rate_In;

private
    task type Rate_Controller
    --# global in out Flow_Rate;
    is
        pragma Priority (10);
    end Rate_Controller;
end Pump;

package body Pump
is
    rc : Rate_Controller;
    Flow_Rate : Base_Types.Integer_Store;

    procedure Put_Start_Button_Pressed
    is
    begin
```

```
        -- TODO: implement event handler
    end Put_Start_Button_Pressed;

    procedure Put_Flow_Rate(Flow_Rate_In : Integer)
    is
    begin
        Flow_Rate.Put(Flow_Rate_In);
    end Put_Flow_Rate;

    task body Rate_Controller
    is
    begin
      loop
         -- implementation
        end loop;
    end Rate_Controller;
end Pump;
```

Listing 4.10: *Port communication translated to SPARK Ada*

4.3 Automatic translator

The ultimate goal is to create translator, which performs translations described in 4.1 and 4.2 automatically. Automatic translator should enable translation of entire model and parts of the model. Initially, translator should support only subset of AADL entities: the system, process, thread, subprogram and port communication. The following functions should be supported:

- data types translation (as described in section 4.1.1)
- threads to tasks translation (as described in 4.1.3)
- single ports translation (based on section 4.1.2)
- subprogram to procedure/function translation (based on section 4.1.4)
- single package translation with system, which contains ports and feature groups (as described in section 4.1.6)
- property set mapping to SPARK Ada package (like in section 4.1.7)

Second step, would be to introduce BLESS support. Which means, add supported BLESS constructs described in section 4.1.8:

- assertions for threads
- pre- and postconditions for subprograms

Recommended way to create translator is to parse AADL models, create Abstract Syntax Tree (AST) and emit code using Visitor pattern. Parser and AST can be generated using ANTLR[4] (Another Tool for Language Recognition) and its grammar development environment ANTLRWorks[5]. ANTLR 4 (with ANTLRWorks 2) enable automatic AST creation and handle left recursion, which makes parser development much easier and faster. Another tool, Xtext[6] can be also used (instead of ANTLR) for parser and AST generator. For emitting code, StringTemplate[7] (template engine for generating code) can be used.

Development should be performed incrementally. From adding translation for the simplest constructs, like data types or single ports, to port communication and BLESS. First step, would be AADL grammar development. It is recommended to initially, specify only part of required AADL subset and then extend it incrementally. In order to do that, AADL Syntax Card[8] might be helpful. During translator development unit testing and Test Driven Development is recommended. Translation schemes can be used as input and expected output of particular test cases. It will help to ensure correctness of translator while working on new features support.

Additionally, automatic translator should work in two modes:

- Ravenscar: as described above, with protected objects and multiple tasks
- Sequential: single-threaded application, without notion of tasks and protected objects

[4] http://www.antlr.org/
[5] http://tunnelvisionlabs.com/products/demo/antlrworks
[6] http://www.eclipse.org/Xtext/index.html
[7] http://www.stringtemplate.org/
[8] https://wiki.sei.cmu.edu/aadl/images/d/d2/AADL_V2.1_Syntax_Card.pdf

Chapter 5

PCA pump prototype implementation and code generation

This chapter describes running SPARK Ada programs on BeagleBoard-xM platform (3.3), implementation details of PCA pump prototype (5.2)) and code generation from simplified AADL/BLESS models of PCA pump (5.3). All programs presented in this section works the same on Intel processor (PC or MacBook) and on BeagleBoard-xM (ARM device).

5.1 Running SPARK Ada programs on BeagleBoard-xM

To run SPARK Ada program on BeagleBoard-xM, it has to be cross-compiled. As an IDE for SPARK Ada development, GNAT Programming Studio (GPS) is used (see section 2.5.2). To create "Hello, World!" application, new Ada project has been created (choosing Project/New... from the menu). Then main.adb file, with procedure Main printing "Hello, World!" in standard output, was added. The code is presented in listing 5.1. It is valid Ada 2005 and Ada 2012 code.

```
with Ada.Text_IO;
```

```
procedure Main
is
begin
    Ada.Text_IO.Put_Line("Hello, World!");
end Main;
```

Listing 5.1: *"Hello World" in Ada*

The main file has to be always specified in project file (`.gpr`) in order to compile and link application, which can be runnable. It can be done in Project/Edit Project Properties (figure 5.1), tab: Main files (figure 5.1) or directly in project file (`.gpr`).

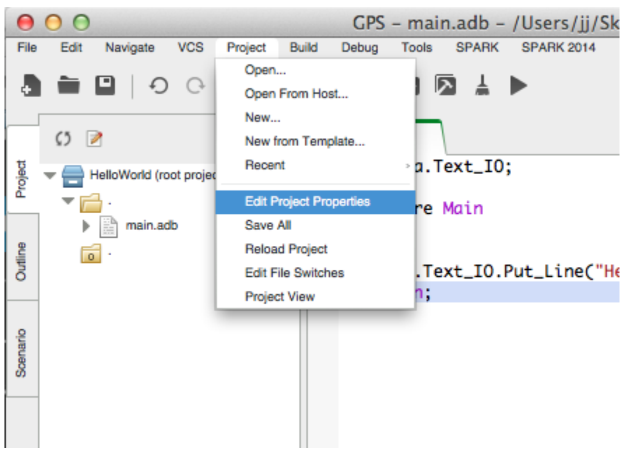

Figure 5.1: *Edit Project Properties*

To enable cross-compilation, for current version of cross-compiler, environmental variable `\$ENV_PREFIX` has to be set to directory, which contains `/lib` and `/usr` directories. Latter should also contains `/usr/lib` and `/usr/include` subdirectories. After mentioned directories has been copied into `/home/super/angstrom-arm` directory, `\$ENV_PREFIX` has been exported with following command: `export ENV_PREFIX=/home/super/angstrom-arm`. Entire project can be compiled and linked with following command: `arm-linux-gnueabi-gnatmake -d -Phelloworld.gpr` (where `helloworld.gpr` is GNAT Programming Studio project file). Additional flags can be specified in command line or directly in project file (manually or through GNAT Programming Studio Interface).

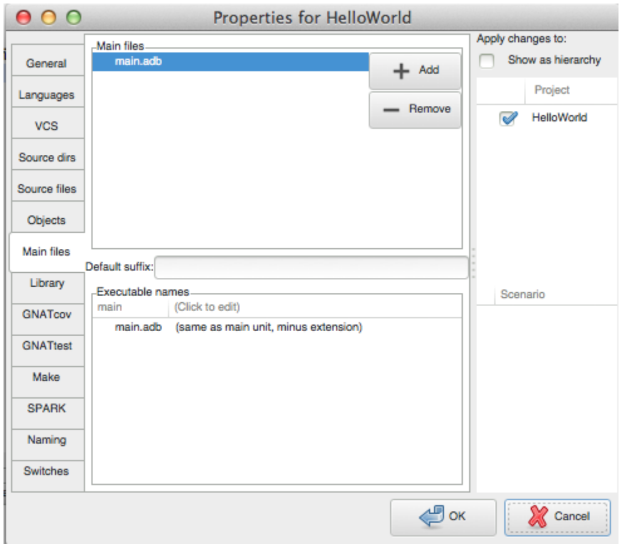

Figure 5.2: *Project Main files*

More complex example, which takes advantage of SPARK contracts is presented in section 5.1.1.

5.1.1 Odometer

Odometer is a simple SPARK Ada program, which implements basic functions of standard Odometer. Listing 5.1.1 shows Odometer in SPARK 2005. In addition to `Odometer`

```
package Odometer
--# own
--# Trip,        -- number of meters so far on this trip (can be reset to 0).
--# Total        -- total meters traveled of vehicle since the last factory-reset.
--#   : Natural; -- has range 0 .. Integer'Last.
--# initializes Trip,
--#             Total;
is
    procedure Zero_Trip; -- sets Trip to 0 and clears all saved Trip marks.
    --# global out Trip;
    --# derives Trip from ;
    --# post Trip = 0;

    function Read_Trip return Natural; -- returns value of Trip.
    --# global in Trip;
    --# return Trip;

    function Read_Total return Natural; -- returns value of Total
    --# global in Total;
    --# return Total;
```

```
    procedure Inc; -- increments each of Trip and Total by 1.
    --# global in out Trip, Total;
    --# derives Trip from Trip & Total from Total;
    --# pre Trip < Integer'Last and Total < Integer'Last;
    --# post Trip = Trip~ + 1 and Total = Total~ + 1;
end Odometer;

package body Odometer is
    Trip : Natural := 0;
    Total : Natural := 0;

    procedure Zero_Trip is
    begin
        Trip := 0;
    end Zero_Trip;

    function Read_Trip return Natural is
    begin
        return Trip;
    end Read_Trip;

    function Read_Total return Natural is
    begin
        return Total;
    end Read_Total;

    procedure Inc is
    begin
        Trip := Trip + 1;
        Total := Total + 1;
    end Inc;
end Odometer;
```

Listing 5.2: *SPARK 2005 code: Odometer*

There are 4 subprograms (2 procedures and 2 functions), which are globally available (through other packages and program units):

- `Zero_Trip` procedure - reset Odometer to 0
- `Read_Trip` function - returns current distance
- `Read_Total` function - returns total distance traveled
- `Inc` procedure - increment total and current distance by 1

Given program contains code contracts. Tough it does not matter in compilation phase, it shows that SPARK verification tools can be used for given example.

Annotation `global` means that subprogram uses some global variable. Postfix in, out or in out means that particular variable is read, write or read and write respectively. Annotation derives says that some variable value depends on other variables. E.g. in procedure Inc variable Trip is dependent on its current value (before procedure call). Annotations pre and post define pre- and postconditions of procedure. We can see, that in Zero_Trip procedure postcondition requires that variable Trip is equal to 0. In procedure Inc, postconditions requires that variables Trip and Total are incremented by 1 ('~' is the value of variable before procedure call). Annotation own expose private variables for use in public methods specification. Annotation initializes ensures that given variables are initializes.

In order to test Odometer package in runtime, the Main procedure has been created. It is presented in listing 5.1.1.

```
with Ada.Text_IO;
with Odometer;
procedure Main
is
begin
    Ada.Text_IO.Put_Line("Trip:  " & Natural'Image(Odometer.Read_Trip));
    Ada.Text_IO.Put_Line("Total: " &  Natural'Image(Odometer.Read_Total));

    Odometer.Inc;

    Ada.Text_IO.Put_Line("Trip:  " & Natural'Image(Odometer.Read_Trip));
    Ada.Text_IO.Put_Line("Total: " &  Natural'Image(Odometer.Read_Total));

    Odometer.Zero_Trip;

    Ada.Text_IO.Put_Line("Trip:  " & Natural'Image(Odometer.Read_Trip));
    Ada.Text_IO.Put_Line("Total: " &  Natural'Image(Odometer.Read_Total));

    Odometer.Inc;

    Ada.Text_IO.Put_Line("Trip:  " & Natural'Image(Odometer.Read_Trip));
    Ada.Text_IO.Put_Line("Total: " &  Natural'Image(Odometer.Read_Total));
end Main;
```

Listing 5.3: *Main procedure for Odometer package*

Odometer in SPARK 2005 works fine on BeagleBoard-xM. In order to test SPARK 2014 program, SPARK 2005 annotations has been converted into Ada 2012 contracts. Listing 5.4 presents Odometer in SPARK 2014.

```
package Odometer
with Abstract_State => (Trip_State, Total_State)
is
   function Trip_State return Integer
     with Convention => Ghost,
     Global => (Input => Trip_State);
```

```
   function Total_State return Integer
     with Convention => Ghost,
     Global => (Input => Total_State);

   procedure Zero_Trip -- sets Trip to 0
     with Global => (Output => (Trip_State)),
     Depends => (Trip_State => null),
     Post => Trip_State = 0;

   function Read_Trip return Natural -- returns value of Trip.
     with Global => (Input => Trip_State),
     Post => Read_Trip'Result = Trip_State;

   function Read_Total return Natural -- returns value of Total
     with Global => (Input => Total_State),
     Post => Read_Total'Result = Total_State;

   procedure Inc -- increments each of Trip and Total by 1.
     with Global => (In_Out => (Trip_State, Total_State)),
     Depends => (Trip_State => Trip_State, Total_State => Total_State),
     Pre => Trip_State < Integer'Last and Total_State < Integer'Last,
     Post => Trip_State = Trip_State'Old + 1 and Total_State = Total_State'Old + 1;
end Odometer;

package body Odometer
with
  Refined_State => (Trip_State => (Trip),
                    Total_State => (Total))
is
   Trip : Natural;
   Total : Natural;

   function Trip_State return Integer
     with Refined_Global => (Input => Trip)
   is
   begin
      return Trip;
   end Trip_State;

   function Total_State return Integer
     with Refined_Global => (Input => Total)
   is
   begin
      return Total;
   end Total_State;

   procedure Zero_Trip
     with Refined_Global => (Output => Trip),
     Refined_Depends => (Trip => null)
   is
   begin
      Trip := 0;
   end Zero_Trip;

   function Read_Trip return Natural
     with Refined_Global => (Input => Trip)
   is
   begin
      return Trip;
    end Read_Trip;

   function Read_Total return Natural
     with Refined_Global => (Input => Total)
   is
```

```
   begin
      return Total;
   end Read_Total;

   procedure Inc
     with Refined_Global => (In_Out => (Trip, Total)),
     Refined_Depends => (Trip => Trip, Total => Total)
   is
   begin
      Trip := Trip + 1;
      Total := Total + 1;
   end Inc;
end Odometer;
```

Listing 5.4: *SPARK 2014 code: Odometer*

Odometer example was created to check possible limitations and issues related to different platform (ARM-based). No limitations were found.

5.1.2 Multitasking applications

In Ada World, concurrency is referred as tasking and the task is the same construct as the thread in other programming languages. In section 5.1.1, single-tasking application was tested. This section presents simple Ada multitasking application and multitasking version of Odometer in SPARK 2005 from section 5.1.1. Both applications compiles correctly and works as expected on BeagleBoard-xM platform.

Ada multitasking application

Listing 5.1.2 presents a simple multitasking application printing numbers in different time intervals, in Ada 2005. It is also valid code for Ada 2012. There are 3 tasks:

- Main task
- S (type: `Seconds`) - simple counter printing numbers form 1 to 10 in every second
- T (type: `Tenth_Seconds`) - simple counter printing numbers from 0.1 to 10 in every 0.1 second

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Float_Text_IO;

procedure Main is
    task type Seconds is
    end Seconds;

    task type Tenth_Seconds is
    end Tenth_Seconds;

    S : Seconds;
    T : Tenth_Seconds;

    task body Seconds is
    begin
        for I in 1..10 loop
            delay Standard.Duration(1);
            Put_Line(Integer'Image(I));
        end loop;
    end Seconds;

    task body Tenth_Seconds is
    begin
        for I in 1..100 loop
            delay 0.1;
            Ada.Float_Text_IO.Put(Float(I)/Float(10), AFT=>2, EXP=>0);
            Put_Line("");
        end loop;
    end Tenth_Seconds;
begin
    Put_Line("Started");
end Main;
```

Listing 5.5: *Simple multitask application in Ada*

The program works as expected on BleagleBoard-xM. This is not valid SPARK program though. As mentioned in section 2.5.3, tasks can be declared only in packages. Not in subprograms or in other tasks. [Bar13]

SPARK Ada multitasking application

As mentioned in section 2.5.3, in SPARK 2005 multitasking is possible with Ravenscar Profile. Default profile - sequential - does not enable tasking. In other words, SPARK tools cannot analyze and reason about programs if Ravenscar profile flag is not provided. In SPARK 2014 - for now tasking is not possible. It's part of SPARK 2014 road map to include support for tasking in the future. Thus, only SPARK 2005 application was tested.

Tested, multitasking application is extended version of Odometer presented in listing 5.1.1. It has additional variable `Speed`, procedure `Set_Speed` and new task: `Drive`. Thus, in total it has two tasks:

- Main
- Drive

The `Drive` task increase `Total` and `Trip` variables by `Speed` (m/s) in every second. Extended Odometer is presented in listing 5.1.2.

```
--# inherit Ada.Real_Time;
package Odometer
--# own Trip : Distance;
--#     Total : Distance;
--#     Speed : Meters_Per_Second;
--#     task d : Drive;
--# initializes Trip,
--#             Total,
--#             Speed;
is
    type Distance is range Natural'First .. Natural'Last;
    pragma Atomic (Distance);

    type Meters_Per_Second is range Natural'First .. Natural'Last;
    pragma Atomic(Meters_Per_Second);
```

```
    procedure Zero_Trip; -- sets Trip to 0 and clears all saved Trip marks.
    --# global out Trip;
    --# derives Trip from ;
    --# post Trip = 0;

    function Read_Trip return Distance; -- returns value of Trip.
    --# global in Trip;
    --# return Trip;

    function Read_Total return Distance; -- returns value of Total
    --# global in Total;
    --# return Total;

    procedure Inc; -- increments each of Trip and Total by 1.
    --# global in out Trip, Total;
    --# derives Trip from Trip & Total from Total;
    --# pre Trip < Distance'Last and Total < Distance'Last;
    --# post Trip = Trip~ + 1 and Total = Total~ + 1;

    procedure Set_Speed(New_Speed : Meters_Per_Second);
    --# global out Speed;
    --# derives Speed from New_Speed;
    --# post Speed = New_Speed;
private
    task type Drive
    --# global in     Speed;
    --#        in out Trip;
    --#        in out Total;
    --#        in     Ada.Real_Time.ClockTime;
    is
        pragma Priority(10);
    end Drive;
end Odometer;

with Ada.Real_Time;
use type Ada.Real_Time.Time;
package body Odometer is
    Trip : Distance := 0;
```

```
Total : Distance := 0;
Speed : Meters_Per_Second := 0;
d : Drive;

procedure Zero_Trip is
begin
    Trip := 0;
end Zero_Trip;

function Read_Trip return Distance is
begin
    return Trip;
end Read_Trip;

function Read_Total return Distance is
begin
    return Total;
end Read_Total;

procedure Inc is
begin
    Trip := Trip + 1;
    Total := Total + 1;
end Inc;

procedure Set_Speed(New_Speed : Meters_Per_Second)
is
begin
    Speed := New_Speed;
end Set_Speed;

task body Drive
is
    Release_Time : Ada.Real_Time.Time;
    Period : constant Integer := 1000; -- update in every second
begin
    loop
        Release_Time := Ada.Real_Time.Clock + Ada.Real_Time.Milliseconds(Period);
        delay until Release_Time;
```

```
            -- each time round, increase Trip and Total
            for I in Meters_Per_Second range 0 .. Speed loop
                Inc;
            end loop;
        end loop;
    end Drive;
  end Odometer;
```

Listing 5.6: *Multitasking Odometer*

There are two ways to access protected variable in task body:

- It has to be protected object
- It has to be atomic type

Protected variables may not be used in proof contexts. Thus, if we try to use protected variable in proofs (pre- or postcondition), then we get semantic error: `Trip is a protected own variable`. To preserve pre- and postconditions from original Odometer, atomic types (`Distance` and `Meters_Per_Second`) has been used. The capability to specify pre- and postconditions has been preserved, but now application is not thread safe.

5.1.3 Controlling PCA pump actuator

PCA pump prototype created as part of this thesis interacts with external device (physical pump) through General-purpose input/output (GPIO) pin. To control the pump, Pulse width modulation (described in 3.3) is used. BeagleBoard-xM has 28 GPIO pins. Three of them are PWM enable (pin 4 - mapped as `GPIO_144`, pin 6 - `GPIO_146` and pin 10 - `GPIO_145`). All of these pins allow to control external device by specifying frequency and duty cycle. However it requires PWM driver[1]. PWM can be also simulated manually. To run the pump, pin has to be turned on and off with specified frequency. In order to do that, `sleep` function can be used.

[1]http://beagleboard.org/project/PWM+driver+for+Beagle+Board/

GPIO ports interact with the BeagleBoard platform through memory maps. This means that turning particular pin on or off is achieved by writing values into a memory segment associated with the pin. Memory segment is further mapped into file system. Memory maps are synchronized via continuous refresh loops.

Pin, used for controlling PCA pump, is the pin 14 (mapped as `GPIO_162`). It is mapped into directory `/sys/class/gpio/gpio162/`. To turn pin on, file `/sys/class/gpio/gpio162/value` has to contain '1'. To turn it off - '0'. Pump is also connected to ground (GND). In that purpose pin 28 is used. Listing 5.1.3 shows simple bash script, which turns pin on and off every second. Before pin can be used, it has to be opened by writing pin mapping number (in this case: 162 for pin 14) into `/sys/class/gpio/export` file. When communication is over, connection should be closed with writing the same value to file `/sys/class/gpio/unexport`. Setting 'high' (1) for 1 second and 'low' (0) also for 1 second gives 50% duty cycle.

```
#!/bin/sh

if [ $# = 0 ]
then
  GPIO=162
else
  GPIO=$1
fi

cleanup() {
  echo $GPIO > /sys/class/gpio/unexport
  exit
}

trap cleanup SIGINT

echo $GPIO > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio$GPIO/direction

while [ "1" = "1" ]; do
  echo "1" > /sys/class/gpio/gpio$GPIO/value
```

```
  sleep 1
  echo "0" > /sys/class/gpio/gpio$GPIO/value
  sleep 1
done

cleanup
```

Listing 5.7: *Turning pin on and off*

Initial tests of interaction with pump actuator has been made in bash and Java, because it does not require cross-compilation. Bash script runs natively on Angstrom Linux. Java application - on JVM distribution for Angstrom.

BeagleBoard-xM with Linux Angstrom allows to install software packages using package manager `opkg`[2]. Packages feeds can be found on `http://feeds.angstrom-distribution.org/feeds` and set in `.conf` files in `/etc/opkg` directory. In this thesis version 2012.01 of Angstrom (with Linux 3.0.14+) has been used and the following feeds:

- `base-feed.conf`: `src/gz base http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a/base`
- `beagleboard-feed.conf`: `src/gz beagleboard http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a/beagleboard`
- `debug-feed.conf`: `src/gz debug http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a/debug`
- `gstreamer-feed.conf`: `src/gz gstreamer http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a/gstreamer`
- `noarch-feed.conf`: `src/gz no-arch http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a/all`

[2] http://wiki.openwrt.org/doc/techref/opkg

- `perl-feed.conf`: `src/gz perl http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a/perl`

- `python-feed.conf`: `src/gz python http://feeds.angstrom-distribution.org/feeds/v2012.05/ipk/eglibc/armv7a /python`

Once, feeds are set, it is recommended to update list of available packages with command: `opkg update`. To update all installed packages, following command has to be used: `opkg upgrade`. To install Java runtime-environment (JVM), the following command can be used: `opkg install openjdk-6-java`. Java Development Kit, which contains Java compiler and allows to compile Java programs on BeagleBoard, can be installed with: `opkg install openjdk-6-jdk`.

Similar program to bash script presented in listing 5.1.3, but working for 20 seconds and terminating, written in Java is presented in listing 5.1.3.

```
import java.io.*;

public class PcaMain {
  public static void main(String[] args) throws IOException, InterruptedException {
    final String GPIO = "162";
    final String BASE_DIR = "/sys/class/gpio";
    WriteToFile(BASE_DIR+"/export", GPIO);
    WriteToFile(BASE_DIR+"/gpio"+GPIO+"/direction", "out");
    for(int i=0; i<10; ++i) {
            WriteToFile(BASE_DIR+"/gpio"+GPIO+"/value", "1");
            Thread.sleep(1000);
      WriteToFile(BASE_DIR+"/gpio"+GPIO+"/value", "0");
            Thread.sleep(1000);
    }
    WriteToFile(BASE_DIR+"/unexport", GPIO);
  }

  public static void WriteToFile(String filename, String content) throws IOException {
    File file = new File(filename);
    if (!file.exists()) {
      file.createNewFile();
    }
```

```
        PrintWriter writer = new PrintWriter(filename, "UTF-8");
        writer.println(content);
        writer.close();
    }
}
```

Listing 5.8: *Turning pin on and off*

Extended program from listing 5.1.3, with procedures to start and stop the pump, written in Ada, is presented in listing 5.1.3.

```
with Ada.Real_Time;
use type Ada.Real_Time.Time;
package Pca_Pump is
   procedure Start_Pump;
   procedure Stop_Pump;
   procedure Run_Pump(N: in Integer);
   procedure Write_Signal(Signal: in Integer);
end Pca_Pump;

with Ada.Strings.Unbounded;
use type Ada.Strings.Unbounded;
with Ada.Text_IO.Unbounded_IO;
use type Ada.Text_IO;

package body Pca_Pump
is
   procedure Start_Pump is
      F    : Ada.Text_IO.File_Type;
      Data : Unbounded_String := To_Unbounded_String("pumping");
      File_Export : Ada.Text_IO.File_Type;
      File_Direction : Ada.Text_IO.File_Type;
   begin
      Create(File_Export, Ada.Text_IO.Out_File, "/sys/class/gpio/export");
      Put_Line(File_Export, "162");
      Close(File_Export);

      Create(File_Direction, Ada.Text_IO.Out_File, "/sys/class/gpio/gpio162/direction");
      Put_Line(File_Direction, "out");
      Close(File_Direction);
```

```
   Create(F, Ada.Text_IO.Out_File, "/home/root/pump_status.txt");
   Unbounded_IO.Put_Line(F, Data);
   Put_Line("Pumping...");
   Close(F);
end Start_Pump;

procedure Stop_Pump is
   F    : Ada.Text_IO.File_Type;
   Data : Unbounded_String := To_Unbounded_String("IDLE");
   File_Unexport : Ada.Text_IO.File_Type;
begin
   Create(File_Unexport, Ada.Text_IO.Out_File, "/sys/class/gpio/unexport");
   Put_Line(File_Unexport, "162");
   Close(File_Unexport);

   Create(F, Ada.Text_IO.Out_File, "/home/root/pump_status.txt");
   Unbounded_IO.Put_Line(F, Data);
   Put_Line("Stopped");
   Close(F);
end Stop_Pump;

procedure Run_Pump(N: in Integer) is
   Interval: constant Ada.Real_Time.Time_Span := Ada.Real_Time.Milliseconds(100);
   Next_Time: Ada.Real_Time.Time;
begin
   Next_Time := Ada.Real_Time.Clock;
   Start_Pump;
   for I in Integer range 1 .. N*1000 loop
Next_Time := Next_Time + Interval;
      Write_Signal(1);
      delay until Next_Time;

      Next_Time := Next_Time + Interval;
      Write_Signal(0);
      delay until Next_Time;
   end loop;
   Stop_Pump;
end Run_Pump;
```

```
  procedure Write_Signal(Signal : in Integer) is
     Filename  : String := "/sys/class/gpio/gpio162/value";
     File : Ada.Text_IO.File_Type;
     Data : Unbounded_String;
  begin
     Ada.Text_IO.Open (File => File,
                       Mode => Ada.Text_IO.Out_File,
                       Name => Filename);
     if Signal = 1 then
        Data := To_Unbounded_String("1");
     else
        Data := To_Unbounded_String("0");
     end if;
     Unbounded_IO.Put_Line(File, Data);
     Ada.Text_IO.Close(File);
  end Write_Signal;
end Pca_Pump;
```

Listing 5.9: *Turning pin on and off*

5.2 Implementation based on Requirements Document and AADL models

In order to confirm that implementation of PCA Pump, specified in Requirements Document, is feasible on BeagleBoard-xM, simple PCA pump prototype has been created. Implemented prototype is multitasking application (using Ravenscar profile) running on BeagleBoard-xM. The base for implementation was `Pca_Operation` package. Only two AADL threads are implemented: `Rate_Controler` and `Max_Drug_Per_Hour_Watcher`. Thus, pump has three tasks in total:

- main task - interface for controlling and monitoring the pump

- `Rate_Controller` - control the speed of infusion rate
- `Max_Drug_Per_Hour_Watcher` - control over infusion

The first step was to translate types required by operation module. Strings and float types were skipped to keep verification simple (using only integer types and its subtypes). Besides that, all types from following packages are translated:

- `Base_Types`
- `Bless_Types`
- `Ice_Types`
- `Pca_Types`

The Open PCA pump, according to requirements document [Lar14], has 5 operational modes:

- Stopped: $F = 0ml/hr$
- Keep Vein Open (KVO): $F = 0.1ml/hr$
- Basal infusion: $F = F_{Basal}$
- Patient bolus: $F = F_{Basal} + F_{Bolus}$
- Clinician bolus: $F = F_{Basal} + F_{Square_Bolus}$ (square bolus is calculated value: VTBI divided by the duration chosen by the clinician)

Requirements document does not specify implementation details. One of implementation decisions, which had to be made, was to decide how basal infusion will work. One solution was to run actuator continuously on speed calculated based on current flow rate. Another solution was to dose drug in 0.1 ml increments. This is how CADD-Prizm Ambulatory

Infusion Pump [Med10] works and this implementation was chosen. It allows for easier bolus monitoring and calculations. Pump engine controller is separated module. It is written in Ada, so it will not be verified with SPARK tools. Using increments, instead of continues speed allows to issue request of 0.1 ml dose to engine module, and it is its responsibility to deliver requested amount of dose. Performing calculations based on speed changes would be much more complicated. For monitoring, amount of drug dosed in last hour (to guard against over infusion), array with size = (60 * 60) has been created. Its elements represents all seconds of last hour. Last element is incremented once request to the engine is issued. This is done in `Rate_Controller` task. `Max_Drug_Per_Hour_Watcher` checks dosed amount by summing all elements. It also shifts array in every second, so doses older than 1 hour are not take into account anymore.

To avoid using floating point types, internal calculations are in micro liters: 1 micro liter (μl) = 0.001 ml, thus 1 ml = 1000μl.

In real-world applications, the embedded critical components are written in SPARK while the non-critical components are written in Ada. Components written in Ada should be hidden for SPARK Examiner with `--# hide` annotation or being separated entities on which SPARK tools are not run. `Pca_Engine` package is separated entity, which control the pump actuator. It use Ada features not present in SPARK, thus it is not verified by SPARK tools.

Implemented PCA pump prototype is console Ada application with textual interface, which has following functionalities:

- Entering prescription, which comprises of following parameters:
  - Basal flow rate
  - Volume to be infused (VTBI) during patient or clinician bolus
  - Maximum dose of drug allowed per hour
  - Minimum time between patient's boluses

- Starting the pump
- Stopping the pump
- Monitoring drug dosed in last hour: when maximum allowed dose is exceeded, it switches pump state to KVO rate
- Performing patient bolus:
  - if bolus request too soon (faster than minimum time between bolus) then it is ignored
  - if bolus is requested during clinician bolus, then clinician bolus is paused and patient bolus starts; once patient bolus is done, pump switches back to clinician bolus
- Performing clinician bolus (time has to be specified):
  - bolus requested during previously requested (not finished) clinician bolus is ignored
  - bolus requested during patient bolus is performed right after patient bolus is done

Code listing of implemented PCA pump along with mapped types is attached in appendix A

5.3 Code generation from AADL/BLESS models

The original AADL/BLESS models were simplified and truncated to demonstrate sample translation. Finally only `PCA_Operation` module with 3 threads (`Max_Drug_Per_Hour_Watcher`, `Rate_Controller`, `Patient_Bolus_Checker`), types definitions (`Base_Types`, `PCA_Types`, `ICE_Types`, `Bless_Types`) and property set `PCA_Properties` were used as the source for code translation. Simplified

AADL/BLESS models can be found in appendix C. The translation was performed based on translation schemes from chapter 4. Appendix D contains translated PCA pump code.

Raw, translated code cannot be verified with SPARK tools, because it contains not implemented parts. E.g. translated from BLESS assertions, which are defined but not implemented in models. Once, these missing parts will be implemented, code can be verified.

[ADD SOME LISTINGS HERE? SHOW WHICH PARTS HAVE TO BE IMPLEMENTED?]

Chapter 6

Verification

The strategy for Software Verification using SPARK tools is as follows. First, Examiner generates and discharge some Verification Conditions (VCs) and Dead Path Conjectures (DPCs). Next, SPARKSimp runs Simplifier to simplify and discharge some (or all) VCs, which were not discharged by Examiner. SPARKSimp runs also ZombieScope to analyze DPCs and ViCToR to discharge VCs (not discharged by Examiner nor Simplifier) with SMT Solver. To get summary of results, POGS report is generated. In case, when not all Verification Conditions are discharged, analysis continues with Bakar Kiasan. After fixes made with Kiasan help, Examiner and SPARKSimp tools are run again to confirm correctness. This approach is presented in the figure 6. Detailed overview of SPARK verification tools can be found in chapter 12 of SPARK book [Bar13].

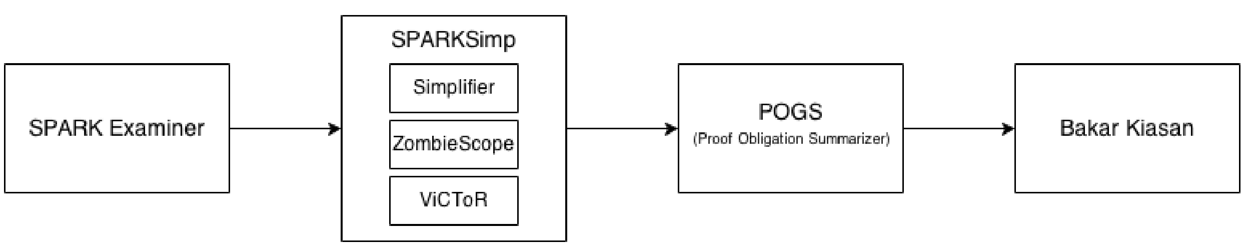

Figure 6.1: *SPARK verification strategy*

6.1 Verification of implemented prototype

During PCA pump prototype implementation, syntax was regularly checked with SPARK Examiner. Complete, manually implemented prototype, which can be found in appendix A, was verified with strategy given at the beginning of this chapter (excluding Bakar Kiasan, which does not handle Ravenscar programs). Thus SPARK Examiner, SPARKSimp (Simplifier, ZombieScope and ViCToR) and POGS were run. The result of this analysis in the form of POGS report summary is presented in listing 6.1. Full report can be found in appendix B.

Three false VCs applies to `Pca_Engine` module, which should not be taken into account during verification, thus they are ignored. 30% (90) of VCs were discharged by Examiner and 60% (183) by Simplifier. There are 29 undischarged VCs. In addition to VCs, DPCs were generated and 32 dead paths were found. Some undischarged VCs and dead paths come from procedures responsible for maximum dose monitoring. As mentioned in chapter 2.6.9, Bakar Kiasan does not support Ravenscar profile. Thus, to be able to analyze monitoring dosed amount of drug, separate, sequential module was created. Verification process of this module is described in section 6.2.

```
Summary:

The following subprograms have VCs proved false:

   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/start_pumping.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/stop_pumping.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/write_signal.vcg

The following subprograms have undischarged VCs (excluding those proved false):

   2  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_time_between_activations.
    vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/get.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/inc.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/put.vcg
   2  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/sum.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/max_drug_per_hour_watcher.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/patientbolus.vcg
  20  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/rate_controller.vcg

Proof strategies used by subprograms
-------------------------------------------------------------------------
```

```
Total subprograms with at least one VC proved by examiner:           15
Total subprograms with at least one VC proved by simplifier:         20
Total subprograms with at least one VC proved by contradiction:       0
Total subprograms with at least one VC proved with user proof rule:   0
Total subprograms with at least one VC proved by Victor:              0
Total subprograms with at least one VC proved by Riposte:             0
Total subprograms with at least one VC proved using checker:          0
Total subprograms with at least one VC discharged by review:          0

Maximum extent of strategies used for fully proved subprograms:
------------------------------------------------------------------------
Total subprograms with proof completed by examiner:                   0
Total subprograms with proof completed by simplifier:                14
Total subprograms with proof completed with user defined rules:       0
Total subprograms with proof completed by Victor:                     0
Total subprograms with proof completed by Riposte:                    0
Total subprograms with proof completed by checker:                    0
Total subprograms with VCs discharged by review:                      0

Overall subprogram summary:
------------------------------------------------------------------------
Total subprograms fully proved:                                      14
Total subprograms with at least one undischarged VC:                  8  <<<
Total subprograms with at least one false VC:                         3  <<<
                                                                  -----
Total subprograms for which VCs have been generated:                 25


ZombieScope Summary:
------------------------------------------------------------------------
Total subprograms for which DPCs have been generated:                25
Total number subprograms with dead paths found:                       3
Total number of dead paths found:                                    32


VC summary:
------------------------------------------------------------------------
Note: (User) denotes where the Simplifier has proved VCs using one or
      more user-defined proof rules.

Total VCs by type:
------------------
                    Total   Examiner Simplifier False    Undisc.
Assert/Post            96         80         12     3          1
Precondition           12          0         12     0          0
Check stmnt.            0          0          0     0          0
Runtime check         187          0        159     0         28
Refinem. VCs           10         10          0     0          0
Inherit. VCs            0          0          0     0          0
================================================================
Totals:               305         90        183     3         29 <<<
%Totals:                         30%        60%    1%        10%
```

Listing 6.1: *Summary of POGS report for PCA Pump prototype*

6.2 Monitoring dosed amount

Verification of module responsible for tracking dosed amount of drug. Isolated to verify, because of Ravenscar limitations. Sequential.

```
package Pca_Pump
--# own Dosed;
--#     Dose_Volume;
--# initializes Dosed,
--#             Dose_Volume;
is
    subtype Integer_Array_Index is Integer range 1 .. 60*60;
    type Integer_Array is array (Integer_Array_Index) of Integer;

    procedure Increase_Dosed;
    --# global in out Dosed;
    --#        in Dose_Volume;
    --# derives Dosed from Dosed, Dose_Volume;

    function Read_Dosed return Integer;
    --# global in Dosed;

    procedure Move_Dosed;
    --# global in out Dosed;
    --# derives Dosed from Dosed;

end Pca_Pump;

package body Pca_Pump
is
    Dosed : Integer_Array := Integer_Array'(others => 0);
    Dose_Volume : Integer := 1;

    procedure Increase_Dosed
    is
    begin
        Dosed(Integer_Array_Index'Last) := Dosed(Integer_Array_Index'Last) + Dose_Volume;
    end Increase_Dosed;

    function Read_Dosed return Integer
    is
        Result : Integer := 0;
    begin
        for I in Integer_Array_Index loop
            --# assert I > 1 -> Result >= Dosed(I-1);
            Result := Result + Dosed(I);
        end loop;
        return Result;
    end Read_Dosed;

    procedure Move_Dosed
    is
    begin
        for I in Integer_Array_Index range 1 .. Integer_Array_Index'Last-1 loop
            --# assert I > 1 -> Dosed(I-1) = Dosed(I);
            Dosed(I) := Dosed(I+1);
        end loop;
        Dosed(Integer_Array_Index'Last) := 0;
    end Move_Dosed;
```

```
end Pca_Pump;
```

Listing 6.2: *Dose monitor module specification*

Verification with Examiner, Simplifier, ZombieScope, Victor, POGS and then Bakar Kiasan. SPARKSimp run Simplifier and Victor with command `sparksimp -victor`.

Examiner: No errors or warnings
[TRUNCATE?] POGS Report:

```
--------------------------------------------------------------------------------
                          Semantic Analysis Summary
                                POGS GPL 2012
             Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
--------------------------------------------------------------------------------

Summary of:

Verification Condition files (.vcg)
Simplified Verification Condition files (.siv)
Victor result files (.vct)
Riposte result files (.rsm)
Proof Logs (.plg)
Dead Path Conjecture files (.dpc)
Summary Dead Path files (.sdp)

"status" column keys:
    1st character:
        '-' - No VC
        'S' - No SIV
        'U' - Undischarged
        'E' - Proved by Examiner
        'I' - Proved by Simplifier by Inference
        'X' - Proved by Simplifier by Contradiction
        'P' - Proved by Simplifier using User Defined Proof Rules
        'V' - Proved by Victor
        'O' - Proved by Riposte
        'C' - Proved by Checker
        'R' - Proved by Review
        'F' - VC is False
    2nd character:
        '-' - No DPC
        'S' - No SDP
        'U' - Unchecked
        'D' - Dead path
        'L' - Live path

in the directory:
/Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification

Summary produced: 01-JUL-2014 14:43:18.04

File /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/increase_dosed.
    vcg
procedure Pca_Pump.Increase_Dosed

VCs generated 01-JUL-2014 14:42:26

VCs simplified 01-JUL-2014 14:43:04
```

```
File /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/increase_dosed.
    dpc
DPCs generated 01-JUL-2014 14:42:26

DPC ZombieScoped 01-JUL-2014  14:43:0

VCs for procedure_increase_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 9        | Undischarged       | Unchecked |   UU   |
| 2   | start |    assert @ finish   | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------


File /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/move_dosed.vcg
procedure Pca_Pump.Move_Dosed

VCs generated 01-JUL-2014 14:42:26

VCs simplified 01-JUL-2014 14:43:04

File /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/move_dosed.dpc
DPCs generated 01-JUL-2014 14:42:26

DPC ZombieScoped 01-JUL-2014  14:43:0

VCs for procedure_move_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 26       | Inference          | Unchecked |   IU   |
| 2   | start | rtc check @ 26       | Inference          | Unchecked |   IU   |
| 3   | start |    assert @ 27       | Inference          | Live      |   IL   |
| 4   | 27    |    assert @ 27       | Inference          | Live      |   IL   |
| 5   | 27    | rtc check @ 28       | Inference          | Unchecked |   IU   |
| 6   | start | rtc check @ 30       | Inference          | Unchecked |   IU   |
| 7   | 27    | rtc check @ 30       | Inference          | Unchecked |   IU   |
| 8   | start |    assert @ finish   | Examiner           | Dead      |   ED   |
| 9   | 27    |    assert @ finish   | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------


File /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/read_dosed.vcg
function Pca_Pump.Read_Dosed

VCs generated 01-JUL-2014 14:42:26

VCs simplified 01-JUL-2014 14:43:05

File /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/read_dosed.dpc
DPCs generated 01-JUL-2014 14:42:26

DPC ZombieScoped 01-JUL-2014  14:43:0

VCs for function_read_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ 17       | Inference          | Live      |   IL   |
| 2   | 17    |    assert @ 17       | Undischarged       | Live      |   UL   |
| 3   | 17    | rtc check @ 18       | Undischarged       | Unchecked |   UU   |
| 4   | 17    |    assert @ finish   | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------
```

```
===============================================================================
Summary:

The following subprograms have undischarged VCs (excluding those proved false):

   1  /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/increase_dosed
     .vcg
   2  /Volumes/External/VMS/shared/aadl-medical/pca-pump-beagleboard/Pca_Verification/pca_pump/read_dosed.vcg

Proof strategies used by subprograms
-------------------------------------------------------------------------
Total subprograms with at least one VC proved by examiner:            2
Total subprograms with at least one VC proved by simplifier:          2
Total subprograms with at least one VC proved by contradiction:       0
Total subprograms with at least one VC proved with user proof rule:    0
Total subprograms with at least one VC proved by Victor:              0
Total subprograms with at least one VC proved by Riposte:             0
Total subprograms with at least one VC proved using checker:          0
Total subprograms with at least one VC discharged by review:          0

Maximum extent of strategies used for fully proved subprograms:
-------------------------------------------------------------------------
Total subprograms with proof completed by examiner:                   0
Total subprograms with proof completed by simplifier:                 1
Total subprograms with proof completed with user defined rules:        0
Total subprograms with proof completed by Victor:                     0
Total subprograms with proof completed by Riposte:                    0
Total subprograms with proof completed by checker:                    0
Total subprograms with VCs discharged by review:                      0

Overall subprogram summary:
-------------------------------------------------------------------------
Total subprograms fully proved:                                       1
Total subprograms with at least one undischarged VC:                  2  <<<
Total subprograms with at least one false VC:                         0
                                                                  -----
Total subprograms for which VCs have been generated:                  3


ZombieScope Summary:
-------------------------------------------------------------------------
Total subprograms for which DPCs have been generated:                 3
Total number subprograms with dead paths found:                        1
Total number of dead paths found:                                     1


VC summary:
-------------------------------------------------------------------------
Note: (User) denotes where the Simplifier has proved VCs using one or
      more user-defined proof rules.

Total VCs by type:
------------------
                    Total   Examiner Simplifier    Undisc.
Assert/Post             8          3          4          1
Precondition            0          0          0          0
Check stmnt.            0          0          0          0
Runtime check           7          0          5          2
Refinem. VCs            0          0          0          0
Inherit. VCs            0          0          0          0
==========================================================
Totals:                15          3          9          3 <<<
%Totals:                         20%        60%        20%
```

```
==================== End of Semantic Analysis Summary =======================
```

Listing 6.3: *POGS report*

pca-pump-verification-step1.png

problem: Integer'First = Integer'Last = 1 :O

solution: added standard.ads:

```
e Standard is

pe Integer is range -2**31 .. 2**31-1;

andard;
```

pca-pump-verification-step2.png

Introduce type Drug_Volume Change `Integer_Array` to `Doses_Array` because it is not array of integers anymore.

Result: no lower overflow in `Increase_Dosed`. Only upper overflow left.

pca-pump-verification-step3.png

Add contract to `Increase_Dosed` `--# pre Read_Dosed(Dosed) <= Drug_Volume'Last - Dose_Volume;` Examiner Error: `Semantic Error   1 - The identifier Read_Dosed is either undeclared or not visible at this point`

.

Moved `Read_Dosed` to be before `Increase_Dosed`. Examiner Error: `pca_pump.ads:19:51: Semantic Error 35 - Binary operator is not declared for types Drug_Volume and Dose_Volume__type.`

Declared `Dose_Volume` type in `--# own`: `--# Dose_Volume : Drug_Volume;`
Rerun Examiner and SPARKSimp: [TRUNCATE?]

```
VCs for procedure_increase_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 9        | Undischarged       | Unchecked |   UU   |
| 2   | start |    assert @ finish   | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------

VCs for procedure_move_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 26       | Inference          | Unchecked |   IU   |
| 2   | start | rtc check @ 26       | Inference          | Unchecked |   IU   |
```

```
| 3   | start |    assert @ 27      | Inference          | Live      |   IL   |
| 4   | 27    |    assert @ 27      | Inference          | Live      |   IL   |
| 5   | 27    | rtc check @ 28      | Inference          | Unchecked |   IU   |
| 6   | start | rtc check @ 30      | Inference          | Unchecked |   IU   |
| 7   | 27    | rtc check @ 30      | Inference          | Unchecked |   IU   |
| 8   | start |    assert @ finish  | Examiner           | Dead      |   ED   |
| 9   | 27    |    assert @ finish  | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------

VCs for function_read_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ 17      | Inference          | Live      |   IL   |
| 2   | 17    |    assert @ 17      | Inference          | Live      |   IL   |
| 3   | 17    | rtc check @ 18      | Undischarged       | Unchecked |   UU   |
| 4   | 17    |    assert @ finish  | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------

===============================================================================
Summary:

Overall subprogram summary:
-------------------------------------------------------------------------
Total subprograms fully proved:                                        1
Total subprograms with at least one undischarged VC:                    2  <<<
Total subprograms with at least one false VC:                           0
                                                                    -----
Total subprograms for which VCs have been generated:                   3


ZombieScope Summary:
-------------------------------------------------------------------------
Total subprograms for which DPCs have been generated:                  3
Total number subprograms with dead paths found:                         1
Total number of dead paths found:                                      1


VC summary:
-------------------------------------------------------------------------
Note: (User) denotes where the Simplifier has proved VCs using one or
      more user-defined proof rules.

Total VCs by type:
------------------
                    Total   Examiner Simplifier    Undisc.
Assert/Post            8          3          5          0
Precondition           0          0          0          0
Check stmnt.           0          0          0          0
Runtime check          7          0          5          2
Refinem. VCs           0          0          0          0
Inherit. VCs           0          0          0          0
============================================================
Totals:               15          3         10          2 <<<
%Totals:                        20%        67%        13%

===================== End of Semantic Analysis Summary ========================
```

Listing 6.4: *Second POGS report*

Now, we can see progress. Only 2 VCs (13%) are undischarged in comparison to 3 (20%) previously.

Then rerun Kiasan.

pca-pump-verification-step4

`Move_Dosed` and `Increase_Dosed` are fine: no Exception cases.

`Read_Dosed` ConstraintError: the value being assigned to Result is too small. After look at the pre and post state it seems weird. After investigation and talk with Kiasan Developer, it was determined that there is a bug in Kiasan v1 (for SPARK 2005). More precisely: checking overflows. For the purpose of verification `Drug_Volume` type range was changed to $0 - (2^{15} - 1)$. Negative values in this case are unnecessary. It will give range up to around 1000000. Which is sufficient even if calculations are made in micro liters (as it is in case of PCA Pump implementation). 1000000 micro liters is 1000 ml, which is 1 liter. Which is extreme amount of drug in case of PCA Pump, according to Requirement Document [LHC13]. The bug with type ranges is fixed in Kiasan v2 (for SPARK 2014).

Another problem is size of Dosed array (3600 elements). First of all, Kiasan array bound and loop bound has to be increased (from default 10). Another thing is computational complexity. The state space grow exponentially and it takes a lot of time to analyze array of 3600 elements. Thus for verification purposes array size was change to 60 elements. Along with increasing array bounds and loop bounds for Kiasan also to 60.

After rerun Kiasan, there is valid test case for `Read_Dose`, but there are also 59 Exception cases: Range violation (UPPER), which means there is possible overflow. One way to fix it is to add `--# assume` annotation to loop in function body, but Kiasan v1 does not support it. Another way is to add pre-condition, which assure, that sum of elements is lower than `Drug_Volume'Last`. SPARK does not provide simple library for summing array (like Contracts for Java provide). Thus, this function has to be implemented. However, its implementation is the same like `Read_Dosed`. It sum all elements of array. Sum function specification and body is presented in listing 6.5.

```
function Sum(Arr : Doses_Array) return Drug_Volume;
```

```
function Sum(Arr : Doses_Array) return Drug_Volume
is
    Result : Drug_Volume := 0;
begin
    for I in Doses_Array_Index loop
        --# assert true;
        Result := Result + Arr(I);
    end loop;
    return Result;
end Sum;
```

Listing 6.5: *Sum function for summing all elements of array*

After rerun Kiasan, there is only valid test case.

pca-pump-verification-step5

The last thing which can be improved by code contracts is checking if `Move_Dosed` procedure works as expected. In that purpose three postconditions were added (listing 6.6). First checks if the last element is equal to 0. Second and third checks two possible scenarios:

- before running procedure, the first element is equal to 0: amount of dosed drug in last hour will not change after Dosed procedure execution
- the first element is greater than 0: after Dosed procedure execution, the amount of drug dosed in last hour will decrease, because first element value will no longer be in last hour range

```
--# post Dosed(Doses_Array_Index'Last) = 0
--#      and (Dosed~(Doses_Array_Index'First)=0 -> Read_Dosed(Dosed~) = Read_Dosed(Dosed))
--#      and (Dosed~(Doses_Array_Index'First)>0 -> Read_Dosed(Dosed~) > Read_Dosed(Dosed));
```

Listing 6.6: *Postconditions added to Move_ Dosed procedure*

After adding these postconditions Kiasan generates 2 test cases to check both mentioned scenarios. There is no error cases, which means that procedure works as expected.

Better way to validate such requirements is Unit testing. In section 6.4, there is overview of unit tests created to test behavior described above.

Running Examiner and SPARKSimp after all changes (truncated result):

```
VCs for procedure_increase_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 20       | Undischarged       | Unchecked |   UU   |
| 2   | start |    assert @ finish   | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------

VCs for procedure_move_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 37       | Inference          | Unchecked |   IU   |
| 2   | start | rtc check @ 37       | Inference          | Unchecked |   IU   |
| 3   | start |    assert @ 38       | Inference          | Live      |   IL   |
| 4   | 38    |    assert @ 38       | Inference          | Live      |   IL   |
| 5   | 38    | rtc check @ 39       | Inference          | Unchecked |   IU   |
| 6   | start | rtc check @ 41       | Inference          | Unchecked |   IU   |
| 7   | 38    | rtc check @ 41       | Inference          | Unchecked |   IU   |
| 8   | start |    assert @ finish   | Inference          | Dead      |   ID   |
| 9   | 38    |    assert @ finish   | Undischarged       | Live      |   UL   |
 -----------------------------------------------------------------------------

VCs for function_read_dosed :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ 28       | Inference          | Live      |   IL   |
| 2   | 28    |    assert @ 28       | Inference          | Live      |   IL   |
| 3   | 28    | rtc check @ 29       | Undischarged       | Unchecked |   UU   |
| 4   | 28    |    assert @ finish   | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------

VCs for function_sum :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ 11       | Inference          | Live      |   IL   |
| 2   | 11    |    assert @ 11       | Inference          | Live      |   IL   |
| 3   | 11    | rtc check @ 12       | Undischarged       | Unchecked |   UU   |
| 4   | 11    |    assert @ finish   | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------

================================================================================
Summary:

Overall subprogram summary:
-------------------------------------------------------------------------
Total subprograms fully proved:                                         0
Total subprograms with at least one undischarged VC:                     4  <<<
Total subprograms with at least one false VC:                            0
                                                                    -----
Total subprograms for which VCs have been generated:                    4


ZombieScope Summary:
-------------------------------------------------------------------------
Total subprograms for which DPCs have been generated:                   4
Total number subprograms with dead paths found:                          1
Total number of dead paths found:                                       1

Total VCs by type:
------------------
                    Total   Examiner Simplifier    Undisc.
```

```
Assert/Post            11          1          9          1
Precondition            0          0          0          0
Check stmnt.            0          0          0          0
Runtime check           8          0          5          3
Refinem. VCs            0          0          0          0
Inherit. VCs            0          0          0          0
=========================================================
Totals:                19          1         14          4 <<<
%Totals:                          5%        74%        21%
```

Listing 6.7: *Third POGS report*

Now, there is 4 undischarged VCs, but total number of generated VCs is 19. In previous runs there was only 15. Thus there is 4 new VCs and 2 of them are undischarged. The reason is introduction of `Sum` function of all subprograms which are using it. To confirm this, look at all undischarged VCs. Which is: 1st VC in `increase_dosed.siv` file (listing 6.8, 9th VC in `move_dosed.siv` file (listing 6.9, 3rd VC in `read_dosed.vcg` file (listing 6.10) and 3rd VC in `sum.vcg` file (listing 6.11). They conform to subprograms: `Increase_Dosed`, `Move_Dosed`, `Read_Dosed` and `Sum` respectively.

[JOIN INTO 1 LISTING?]

```
procedure_increase_dosed_1.
H1:    read_dosed(dosed) <= 32767 - dose_volume .
H2:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 60 -> 0 <= element(
          dosed, [i___1]) and element(dosed, [i___1]) <= 32767) .
H3:    dose_volume >= 0 .
H4:    dose_volume <= 32767 .
H5:    integer__size >= 0 .
H6:    drug_volume__size >= 0 .
H7:    drug_volume__base__first <= drug_volume__base__last .
H8:    doses_array_index__size >= 0 .
H9:    drug_volume__base__first <= 0 .
H10:   drug_volume__base__last >= 32767 .
       ->
C1:    element(dosed, [60]) + dose_volume <= 32767 .
```

Listing 6.8: *Undischarged Verification Condition from increase_ dosed.siv file*

```
procedure_move_dosed_9.
H1:    element(dosed, [58]) = element(dosed, [59]) .
H2:    for_all(i___1 : integer, 1 <= i___1 and i___1 <= 60 -> 0 <= element(
          dosed, [i___1]) and element(dosed, [i___1]) <= 32767) .
H3:    element(dosed, [60]) >= 0 .
H4:    element(dosed, [60]) <= 32767 .
H5:    integer__size >= 0 .
H6:    drug_volume__size >= 0 .
H7:    drug_volume__base__first <= drug_volume__base__last .
H8:    doses_array_index__size >= 0 .
H9:    drug_volume__base__first <= 0 .
H10:   drug_volume__base__last >= 32767 .
```

```
        ->
C1:     element(dosed~, [1]) = 0 -> read_dosed(dosed~) = read_dosed(update(
           update(dosed, [59], element(dosed, [60])), [60], 0)) .
C2:     element(dosed~, [1]) > 0 -> read_dosed(dosed~) > read_dosed(update(
           update(dosed, [59], element(dosed, [60])), [60], 0)) .
```

Listing 6.9: *Undischarged Verification Condition from move_ dosed.siv file*

```
function_read_dosed_3.
H1:     loop__1__i > 1 -> result >= element(dosed, [loop__1__i - 1]) .
H2:     for_all(i___1 : integer, 1 <= i___1 and i___1 <= 60 -> 0 <= element(
           dosed, [i___1]) and element(dosed, [i___1]) <= 32767) .
H3:     sum(dosed) <= 32767 .
H4:     loop__1__i >= 1 .
H5:     loop__1__i <= 60 .
H6:     result >= 0 .
H7:     result <= 32767 .
H8:     integer__size >= 0 .
H9:     drug_volume__size >= 0 .
H10:    drug_volume__base__first <= drug_volume__base__last .
H11:    doses_array_index__size >= 0 .
H12:    drug_volume__base__first <= 0 .
H13:    drug_volume__base__last >= 32767 .
        ->
C1:     result + element(dosed, [loop__1__i]) <= 32767 .
```

Listing 6.10: *Undischarged Verification Condition from read_ dosed.siv file*

```
function_sum_3.
H1:     for_all(i___1 : integer, 1 <= i___1 and i___1 <= 60 -> 0 <= element(arr,
           [i___1]) and element(arr, [i___1]) <= 32767) .
H2:     loop__1__i >= 1 .
H3:     loop__1__i <= 60 .
H4:     result >= 0 .
H5:     result <= 32767 .
H6:     integer__size >= 0 .
H7:     drug_volume__size >= 0 .
H8:     drug_volume__base__first <= drug_volume__base__last .
H9:     doses_array_index__size >= 0 .
H10:    drug_volume__base__first <= 0 .
H11:    drug_volume__base__last >= 32767 .
        ->
C1:     result + element(arr, [loop__1__i]) <= 32767 .
```

Listing 6.11: *Undischarged Verification Condition from sum.siv file*

In `Move_Dosed` procedure, tools cannot prove implications in post conditions. Fortunately, it is already proved by Bakar Kiasan. The problem in `Increase_Dosed`, `Read_Dosed` and `Sum` is the same. Tools cannot verify, that adding `Result` and some element of `Dosed` array will not cause overflow. Bakar Kiasan can prove correctness of `Increase_Dosed` and `Read_Dosed`. However only, with assumption that `Sum` is correct. `Sum` cannot be proved by Bakar Kiasan. Four exception cases indicating possible overflow are generated. Thus, the only way to prove correctness of this module is to assume, that helper function `Sum` is correct.

In procedure `Move_Dosed`, there is one dead path found. POGS report gives only information where dead path exists, but not in which circumstances. The information about conditions, in which dead path occurs is stored in `.dpc` file. The file path to concrete file is given in the POGS report just before summary table for procedure `Move_Dosed`. In this case it is `move_dosed.dpc` file. Listing 6.2 presents truncated POGS report, but as an example, full POGS report of implemented PCA prototype can be found in appendix B (e.g. see line 50, which contains DPC analysis for `Start_Pumping` procedure).

Relevant fragment, which applies to found dead path is presented in listing 6.2. It is a list of hypothesis, in which hypothesis 10 (H10) states that number of elements in `Doses_Array` is 1 or less. In this case (or more precisely: in this path), for loop will not be visited. `Doses_Array` has always 3600 elements, thus this path is impossible (dead). It does not mean something bad, because dead path indicate possible issues. In this case it is not issue. It is expected behavior.

```
procedure_move_dosed_8.
H1:    for_all(i___1: integer, ((i___1 >= doses_array_index__first) and (
           i___1 <= doses_array_index__last)) -> ((element(
           dosed, [i___1]) >= drug_volume__first) and (element(
           dosed, [i___1]) <= drug_volume__last))) .
H2:    doses_array_index__last - 1 >= integer__first .
H3:    doses_array_index__last - 1 <= integer__last .
H4:    doses_array_index__last - 1 >= integer__base__first .
H5:    doses_array_index__last - 1 <= integer__base__last .
H6:    doses_array_index__first >= integer__first .
H7:    doses_array_index__first <= integer__last .
H8:    (doses_array_index__first <= doses_array_index__last - 1) -> ((
           doses_array_index__last - 1 >= doses_array_index__first) and (
           doses_array_index__last - 1 <= doses_array_index__last)) .
H9:    (doses_array_index__first <= doses_array_index__last - 1) -> ((
           doses_array_index__first >= doses_array_index__first) and (
           doses_array_index__first <= doses_array_index__last)) .
H10:   not (doses_array_index__first <= doses_array_index__last - 1) .
H11:   0 >= drug_volume__first .
H12:   0 <= drug_volume__last .
H13:   doses_array_index__last >= doses_array_index__first .
H14:   doses_array_index__last <= doses_array_index__last .
        ->
C1:    false .
```

Listing 6.12: *Dead path in `Move_Dosed` procedure*

Complete code of module for dose monitoring can be found in E.

Unfortunately, introduced changes (pre- and postconditions) cannot be applied to PCA Pump prototype implementation, because - as mentioned in chapter 2.6 - protected objects cannot be used in proof annotations (pre- and postconditions).

This shows, how code implemented based on translation from AADL/BLESS can be verified using SPARK tools.

6.3 Verification of generated code

Code translated from AADL models is presented in appendix D. Verification with Examiner of package `Pca_Operation` specification returns syntax error: `Neither KNOWN_DISCRIMINANT_PART nor TASK_TYPE_ANNOTATION can start with reserved word "IS"`. It means, that discriminants or task annotation are expected here. In order to pass Examiner syntax check at least one annotation has to be declared. For demonstration purposes, `Ada.Real_Time.ClockTime` is used. Complete task declaration is presented in listing 6.3.

```
task type Patient_Bolus_Checker
--# global in Ada.Real_Time.ClockTime;
--# derives null from Ada.Real_Time.ClockTime;
is
    pragma Priority(10);
end Patient_Bolus_Checker;
```

Listing 6.13: *Undischarged Verification Condition from sum.siv file*

Once annotation is added, `Pca_Operation` package specification passes Examiner syntax check. Verification of package body returns errors, which are caused by not implemented assertions (translated from BLESS). When all assertions are removed, only flow errors (presented in listing 6.3) are found by Examiner.

```
pca_operation.adb:82:9: Flow Error  30 - The variable Infusion_Flow_Rate is imported but neither referenced
    nor exported.
pca_operation.adb:92:9: Flow Error  30 - The variable Bolus_Duration is imported but neither referenced nor
    exported.
pca_operation.adb:92:9: Flow Error  32 - The variable Infusion_Flow_Rate is neither imported nor defined.
```

```
pca_operation.adb:92:9: Flow Error  31 - The variable Infusion_Flow_Rate is exported but not (internally)
    defined.
pca_operation.adb:92:9: Flow Error  32 - The variable System_Status is neither imported nor defined.
pca_operation.adb:92:9: Flow Error  31 - The variable System_Status is exported but not (internally) defined.
pca_operation.adb:92:9: Flow Error  30 - The variable Rx is imported but neither referenced nor exported.
pca_operation.adb:92:9: Warning 400 - Variable la is declared but not used.
pca_operation.adb:101:9: Flow Error  35 - Importation of the initial value of variable Ada.Real_Time.ClockTime
     is ineffective.
```

Listing 6.14: *Flow errors returned by Examiner for `Pca_Operation` package body*

This is nice indication what has to be implemented in particular parts of the program. It is recommended to not use VC and DPC generation until there are some syntax errors. When all errors are fixed, program can be initially verified as described in previous sections.

6.4 AUnit tests

Better way to prove expected behavior of `Move_Dosed` in Dose monitoring module is to create AUnit test. To check both behaviors of `Move_Dosed` procedure, two tests have been created:

- `Test_Move_Dosed_First_Element_Zero` - first element is 0, then after execution of the procedure dosed amount of drug should be not changed
- `Test_Move_Dosed_First_Element_Not_Zero` - first element is greater than 0, then after execution of the procedure dosed amount of drug should be smaller than before

Both test cases are presented in listing 6.4.

```
procedure Test_Move_Dosed_First_Element_Zero (Gnattest_T : in out Test) is
  pragma Unreferenced (Gnattest_T);
  Pre_Sum : Pca_Pump.Drug_Volume := 0;
  Post_Sum : Pca_Pump.Drug_Volume := 0;
begin
  -- Arrange
  Pre_Sum := Pca_Pump.Read_Dosed;

  -- Act
  Pca_Pump.Move_Dosed;
  Post_Sum := Pca_Pump.Read_Dosed;
```

```
  -- Assert
  AUnit.Assertions.Assert
    (Post_Sum = Pre_Sum,
     "Total dose changed: " &  Pca_Pump.Drug_Volume'Image(Pre_Sum) & " /= " &  Pca_Pump.Drug_Volume'Image(
     Post_Sum));
end Test_Move_Dosed_First_Element_Zero;

procedure Test_Move_Dosed_First_Element_Not_Zero (Gnattest_T : in out Test) is
  pragma Unreferenced (Gnattest_T);
  Pre_Sum : Pca_Pump.Drug_Volume := 0;
  Post_Sum : Pca_Pump.Drug_Volume := 0;
begin
  -- Arrange
  Pca_Pump.Increase_Dosed;
  for I in Pca_Pump.Doses_Array_Index range 1 .. Pca_Pump.Doses_Array_Index'Last-1 loop
     Pca_Pump.Move_Dosed;
  end loop;
  Pre_Sum := Pca_Pump.Read_Dosed;

  -- Act
  Pca_Pump.Move_Dosed;
  Post_Sum := Pca_Pump.Read_Dosed;

  -- Assert
  AUnit.Assertions.Assert
    (Post_Sum < Pre_Sum,
     "Total dose changed: " &  Pca_Pump.Drug_Volume'Image(Pre_Sum) & " should be greater than " &  Pca_Pump.
     Drug_Volume'Image(Post_Sum));
end Test_Move_Dosed_First_Element_Not_Zero;
```

Listing 6.15: *AUnit tests for Move_ Dosed procedure*

6.5 gnatPROVE

The sequential module for monitoring dosed amount verification presented in section 6.2 has been converted to SPARK 2014. In this purpose "SPARK 2005 to 2014" translator created by AdaCore has been used. Translated code is presented in listing .... It use abstract refinement, which is different than in SPARK 2005. To access private, global variables, ghost functions `Dosed_State` and `Dose_Volume_State` have to be used.

Code presented in listing ... passes flow analysis. [START FROM HERE]

In SPARK 2014, the `Standard.ads` file with type ranges is not necessary, because it is handled by language.

Chapter 7

Summary

What I have done.

The work is done for SPARK 2005. SPARK 2014 and its tools (such as gnatPROVE) were not ready at the time, when this thesis was written.

The biggest challenge during PCA pump development was the SPARK limitations. There are many common libraries, which cannot be verified by SPARK tools. Thus it is required to isolate some functionalities or implement them in different way. Another issue was lack of many resources and SPARK code samples. Most of them are used in research or are protected by intelectual property laws.

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 - AADL models don't contains subprograms, which would be useful in AADL2SPARK translator creation

Chapter 8

Future work

The following are possible extensions for work done in this thesis:

- Create automatic translator described in section 4.3.
- Develop translations for BLESS state machine (states and transitions). Good point to start is `Rate_Controller` thread, which can be found in `PCA_Operation_Threads` package in original AADL models created by Brian Larson. The semantics of BLESS contain notions of time that make translation to SPARK difficult. This problem occurs in state machine models. Finding solution for that is needed. Maybe even, by changing BLESS semantics.
- For the time, when this thesis was written, SPARK 2014 did not support multitasking. However, there were plans to introduce it into SPARK 2014 like it took place in case of SPARK 2005. Once, multitasking support would be present, translations for SPARK 2014 will be possible.
- Data types translation presented in section 4.1.1, in additions to straightforward type mapping, comprises of protected types. However, all protected types has the same set

of subprograms (`Put` and `Get`). It is worth to consider introduction of generics, which will allow to specify generic protected type and then reuse it for all types.

- In feature groups translation (section 4.1.5), idea of child or nested packages is abandoned. However, it would be good to reconsider it. Maybe by introduction of getter functions in parent package or some other techniques, which will allow for better separation and decomposition.

- AADL property set mapping in section 4.1.7 handles only `aadlinteger` type. Thus, it requires extension for handling other, more complex constructs.

- Current translation schemes cause creation of pretty big packages, which will become bigger after adding implementation. Thus, some decomposition is desired. Following techniques can be considered:

  - partition of packages
  - take advantage of child packages
  - separation of threads to different packages (e.g. one thread per child package and all common functionalities in parent package)

  simple package separation

- Created PCA pump prototype contains only basic functionalities. Some parameters (like drug concentration) are ignored. The next step is its development, would be taken skipped parts into account. In addition to that, interaction with external modules, like sensors for monitoring drug flow, or communication with ICE through Ethernet port is desired. It requires creation of communication channel between BeagleBoard (SPARK Ada application) and these systems.

- Port communication presented in section 4.2 captures only 1:1 connections between ports of the same type and opposite direction. In AADL there are also inter-port connections and one-to-many or many-to-one connections. [FG13] They should be taken into AADL subset for medical devices modeling and translation.

- Currently AADL thread properties are not take into account in thread to task mapping, in section 4.1.3. Properties like priority or period would be very useful in SPARK Ada programs. For now, former is hard-coded as 10 and latter simply skipped, which requires developer to handle it. However, such property modeled and analyzed in AADL models, should be translated automatically to maintain synchronization between model and the code. AADL properties are described in [FG13], in Appendix A.

- There is an issue with two way communication between SPARK packages caused by circular dependency. It is described in section 4.2.1.

Bibliography

[AB04] Tullio Vardanega Alan Burns, Brian Dobbing. Guide for the use of the ada ravenscar profile in high integrity systems. *ACM SIGAda Ada Letters*, 24(2):1–74, Juin 2004.

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

[AL14a] AdaCore and Altran UK Ltd. Spark 2014 reference manual. URL: `http://docs.adacore.com/spark2014-docs/html/lrm`, 2011-2014.

[AL14b] AdaCore and Altran UK Ltd. Spark 2014 toolset user's guide. URL: `http://docs.adacore.com/spark2014-docs/html/ug`, 2011-2014.

[AW01] Neil Audsley and Andy Wellings. Issues with using ravenscar and the ada distributed systems annex for high-integrity systems. In *IRTAW '00 Proceedings of the 10th international workshop on Real-time Ada workshop*, pages 33 – 39. ACM New York, NY, USA, 2001.

[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.

[Cha00] Roderick Chapman. Industrial experience with spark. *ACM SIGAda Ada Letters - special issue on presentations from SIGAda 2000*, XX(4):64–68, Décembre 2000.

[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.

[DRH07] Xianghua Deng, Robby, and John Hatcliff. Kiasan/kunit: Automatic test case generation and analysis feedback for open object-oriented systems. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, pages 3–12. IEEE Computer Society Washington, DC, 2007.

[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.

[HKL+12] John Hatcliff, Andrew King, Insup Lee, Alasdair MacDonald, Anura Fernando, Michael Robkin, Eugene Vasserman, Sandy Weininger, and Julian M. Goldman. Rationale and architecture principles for medical application platforms. In *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on*, pages 3 – 12. IEEE, 2012.

[HLW12] John Hatcliff, Kejia Li, and Steve Warren. Component-based app design for platform-oriented devices in a medical device coordination framework. In *ACM SIGHIT International Health Informatics Symposium*, pages 343 – 352, 2012.

[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.

[Lev12] Nancy G. Leveson. *Engineering a Safer World.* The MIT Press, 2012.

[LHC13] Brian R. Larson, John Hatcliff, and Patrice Chalin. Open source patient-controlled analgesic pump requirements documentation. In *Software Engineering in Health Care (SEHC), 2013 5th International Workshop*, pages 28–34. Institute of Electrical and Electronics Engineers (IEEE), 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.

[Med10] Smiths Medical. Cadd-prizm ambulatory infusion pump model 6100 and model 6101 - technical manual. URL: http://www.smiths-medical.com/upload/products/pdf/cadd_prizm_vip_system/in19824.pdf, Novembre 2010.

[PHR] Sam Procted, John Hatcliff, and Robby. Towards an aadl-based definition of app architecture for medical application platforms. In *Proceedings of the 2014 Software Engineering in Health-care (SEHC) Workshop at the International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2014)*.

[SC12] Loren Segal and Patrice Chalin. A comparison of intermediate verification languages: Boogie and sireum/pilar. In *Verified Software: Theories, Tools, Experiments*, pages 130–145. Springer, 2012.

[SCD14] 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 draft 0.9, Avril 2014.

[Tea] SPARK Team. Victor wrapper user manual.

[Tea10] SPARK Team. Sparksimp utility user manual. URL: http://docs.adacore.com/sparkdocs-docs/SPARKSimp_UM.htm, Novembre 2010.

[Tea11a] SPARK Team. Pogs user manual. URL: http://docs.adacore.com/sparkdocs-docs/Pogs_UM.htm, Septembre 2011.

[Tea11b] SPARK Team. Spark examiner user manual. URL: http://docs.adacore.com/sparkdocs-docs/Examiner_UM.htm, Décembre 2011.

[Tea11c] SPARK Team. Spark simplifier user manual. URL: http://docs.adacore.com/sparkdocs-docs/Simplifier_UM.htm, Juin 2011.

[Tea12] SPARK Team. The spark ravenscar profile. URL: http://docs.adacore.com/sparkdocs-docs/Examiner_Ravenscar.htm, 2012.

[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, implemented, working pump

Content of this appendix.

Appendix B

PCA pump prototype - simple, implemented, working pump

```
-------------------------------------------------------------------------------
                          Semantic Analysis Summary
                                POGS GPL 2012
            Copyright (C) 2012 Altran Praxis Limited, Bath, U.K.
-------------------------------------------------------------------------------

Summary of:

Verification Condition files (.vcg)
Simplified Verification Condition files (.siv)
Victor result files (.vct)
Riposte result files (.rsm)
Proof Logs (.plg)
Dead Path Conjecture files (.dpc)
Summary Dead Path files (.sdp)

"status" column keys:
    1st character:
        '-' - No VC
        'S' - No SIV
        'U' - Undischarged
        'E' - Proved by Examiner
        'I' - Proved by Simplifier by Inference
        'X' - Proved by Simplifier by Contradiction
        'P' - Proved by Simplifier using User Defined Proof Rules
        'V' - Proved by Victor
        'O' - Proved by Riposte
        'C' - Proved by Checker
        'R' - Proved by Review
        'F' - VC is False
    2nd character:
        '-' - No DPC
        'S' - No SDP
        'U' - Unchecked
```

```
       'D' - Dead path
       'L' - Live path

in the directory:
/Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar

Summary produced: 24-JUL-2014 11:40:08.21

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/start_pumping.vcg
procedure Pca_Engine.Start_Pumping

VCs generated 21-JUL-2014 21:52:42

VCs simplified 24-JUL-2014 11:39:16

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/start_pumping.dpc
DPCs generated 21-JUL-2014 21:52:42

DPC ZombieScoped 21-JUL-2014  21:54:5

VCs for procedure_start_pumping :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish   | False              | Live      |   FL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/stop_pumping.vcg
procedure Pca_Engine.Stop_Pumping

VCs generated 21-JUL-2014 21:52:42

VCs simplified 24-JUL-2014 11:39:16

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/stop_pumping.dpc
DPCs generated 21-JUL-2014 21:52:42

DPC ZombieScoped 21-JUL-2014  21:54:5

VCs for procedure_stop_pumping :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish   | False              | Live      |   FL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/write_signal.vcg
procedure Pca_Engine.Write_Signal

VCs generated 21-JUL-2014 21:52:42

VCs simplified 24-JUL-2014 11:39:16

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/write_signal.dpc
DPCs generated 21-JUL-2014 21:52:42

DPC ZombieScoped 21-JUL-2014  21:54:5

VCs for procedure_write_signal :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish   | False              | Live      |   FL   |
```

```
 ----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/clinicianbolus.vcg
procedure Pca_Operation.ClinicianBolus

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:16

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/clinicianbolus.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_clinicianbolus :
 ----------------------------------------------------------------------------
| #    | From  | To                   | Proved By          | Dead Path | Status |
|----------------------------------------------------------------------------
| 1    | start | rtc check @ 198      | Inference          | Unchecked |   IU   |
| 2    | start | rtc check @ 200      | Inference          | Unchecked |   IU   |
| 3    | start | rtc check @ 201      | Inference          | Unchecked |   IU   |
| 4    | start | rtc check @ 203      | Inference          | Unchecked |   IU   |
| 5    | start |    assert @ finish   | Examiner           | Live      |   EL   |
| 6    | start |    assert @ finish   | Examiner           | Live      |   EL   |
| 7    | start |    assert @ finish   | Examiner           | Live      |   EL   |
 ----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_state.vcg
function Pca_Operation.Get_State

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:16

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_state.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_get_state :
 ----------------------------------------------------------------------------
| #    | From  | To                   | Proved By          | Dead Path | Status |
|----------------------------------------------------------------------------
| 1    | start | rtc check @ 110      | Inference          | Unchecked |   IU   |
| 2    | start |    assert @ finish   | Inference          | Live      |   IL   |
 ----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_time_between_activations.vcg
function Pca_Operation.Get_Time_Between_Activations

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:16

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_time_between_activations.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_get_time_between_activations :
 ----------------------------------------------------------------------------
| #    | From  | To                   | Proved By          | Dead Path | Status |
```

```
|---------------------------------------------------------------------------
| 1   | start | rtc check @ 91      | Undischarged       | Unchecked |   UU   |
| 2   | start | rtc check @ 92      | Inference          | Unchecked |   IU   |
| 3   | start | rtc check @ 95      | Undischarged       | Unchecked |   UU   |
| 4   | start |    assert @ finish  | Inference          | Live      |   IL   |
 ---------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_volume_infused.vcg
function Pca_Operation.Get_Volume_Infused

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:17

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_volume_infused.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_get_volume_infused :
 ---------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|---------------------------------------------------------------------------
| 1   | start |    assert @ finish  | Inference          | Live      |   IL   |
 ---------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/get.vcg
function Pca_Operation.Integer_Array_Store.Get

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:17

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/get.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_get :
 ---------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|---------------------------------------------------------------------------
| 1   | start | rtc check @ 39      | Undischarged       | Unchecked |   UU   |
| 2   | start |    assert @ finish  | Inference          | Live      |   IL   |
| 3   |       | refinement          | Examiner           | No DPC    |   E-   |
| 4   |       | refinement          | Examiner           | No DPC    |   E-   |
 ---------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/inc.vcg
procedure Pca_Operation.Integer_Array_Store.Inc

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:17

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/inc.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_inc :
 ---------------------------------------------------------------------------
```

```
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 56      | Undischarged       | Unchecked |   UU   |
| 2   | start |    assert @ finish  | Examiner           | Live      |   EL   |
| 3   |       | refinement          | Examiner           | No DPC    |   E-   |
| 4   |       | refinement          | Examiner           | No DPC    |   E-   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/pulse.vcg
procedure Pca_Operation.Integer_Array_Store.Pulse

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:17

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/pulse.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_pulse :
 -----------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 76      | Inference          | Unchecked |   IU   |
| 2   | start | rtc check @ 76      | Inference          | Unchecked |   IU   |
| 3   | start |    assert @ 77      | Inference          | Live      |   IL   |
| 4   | 77    |    assert @ 77      | Inference          | Live      |   IL   |
| 5   | 77    | rtc check @ 78      | Inference          | Unchecked |   IU   |
| 6   | start | rtc check @ 80      | Inference          | Unchecked |   IU   |
| 7   | 77    | rtc check @ 80      | Inference          | Unchecked |   IU   |
| 8   | start |    assert @ finish  | Examiner           | Dead      |   ED   |
| 9   | 77    |    assert @ finish  | Examiner           | Live      |   EL   |
| 10  |       | refinement          | Examiner           | No DPC    |   E-   |
| 11  |       | refinement          | Examiner           | No DPC    |   E-   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/put.vcg
procedure Pca_Operation.Integer_Array_Store.Put

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:18

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/put.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_put :
 -----------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 48      | Undischarged       | Unchecked |   UU   |
| 2   | start |    assert @ finish  | Examiner           | Live      |   EL   |
| 3   |       | refinement          | Examiner           | No DPC    |   E-   |
| 4   |       | refinement          | Examiner           | No DPC    |   E-   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/sum.vcg
function Pca_Operation.Integer_Array_Store.Sum
```

```
VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:18

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/sum.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_sum :
 ----------------------------------------------------------------------------
| #    | From  | To                   | Proved By          | Dead Path | Status |
|----------------------------------------------------------------------------
| 1    | start |     assert @ 65      | Inference          | Live      |   IL   |
| 2    | 65    |     assert @ 65      | Undischarged       | Live      |   UL   |
| 3    | 65    | rtc check @ 66       | Undischarged       | Unchecked |   UU   |
| 4    | 65    |     assert @ finish  | Inference          | Live      |   IL   |
| 5    |       | refinement           | Examiner           | No DPC    |   E-   |
| 6    |       | refinement           | Examiner           | No DPC    |   E-   |
 ----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/max_drug_per_hour_watcher.vcg
task_type Pca_Operation.Max_Drug_Per_Hour_Watcher

VCs generated 24-JUL-2014 11:39:13

VCs simplified 24-JUL-2014 11:39:18

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/max_drug_per_hour_watcher.dpc
DPCs generated 24-JUL-2014 11:39:13

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for task_type_max_drug_per_hour_watcher :
 ----------------------------------------------------------------------------
| #    | From  | To                   | Proved By          | Dead Path | Status |
|----------------------------------------------------------------------------
| 1    | start |     assert @ 330     | Examiner           | Live      |   EL   |
| 2    | 330   |     assert @ 330     | Examiner           | Live      |   EL   |
| 3    | 330   |     assert @ 330     | Examiner           | Live      |   EL   |
| 4    | 330   | rtc check @ 332      | Inference          | Unchecked |   IU   |
| 5    | 330   | rtc check @ 333      | Inference          | Unchecked |   IU   |
| 6    | 330   | rtc check @ 333      | Undischarged       | Unchecked |   UU   |
| 7    | 330   | rtc check @ 336      | Inference          | Unchecked |   IU   |
| 8    | 330   | rtc check @ 337      | Inference          | Unchecked |   IU   |
| 9    | 330   | rtc check @ 338      | Inference          | Unchecked |   IU   |
| 10   | 330   | rtc check @ 339      | Inference          | Unchecked |   IU   |
| 11   | 330   |     assert @ finish  | Examiner           | Dead      |   ED   |
| 12   | 330   |     assert @ finish  | Examiner           | Dead      |   ED   |
 ----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_get_basal_flow_rate.vcg
function Pca_Operation.Panel_Get_Basal_Flow_Rate

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:19

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_get_basal_flow_rate.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1
```

```
VCs for function_panel_get_basal_flow_rate :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish   | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_get_max_drug_per_hour.vcg
function Pca_Operation.Panel_Get_Max_Drug_Per_Hour

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:19

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_get_max_drug_per_hour.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_panel_get_max_drug_per_hour :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish   | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/
    panel_get_minimum_time_between_bolus.vcg
function Pca_Operation.Panel_Get_Minimum_Time_Between_Bolus

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:19

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/
    panel_get_minimum_time_between_bolus.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_panel_get_minimum_time_between_bolus :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish   | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_get_vtbi.vcg
function Pca_Operation.Panel_Get_Vtbi

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:19

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_get_vtbi.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for function_panel_get_vtbi :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
```

```
|-----------------------------------------------------------------------------
| 1   | start |    assert @ finish  | Inference          | Live      |   IL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_set_basal_flow_rate.vcg
procedure Pca_Operation.Panel_Set_Basal_Flow_Rate

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:19

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_set_basal_flow_rate.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_panel_set_basal_flow_rate :
 -----------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 117     | Inference          | Unchecked |   IU   |
| 2   | start |    assert @ finish  | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_set_max_drug_per_hour.vcg
procedure Pca_Operation.Panel_Set_Max_Drug_Per_Hour

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:19

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_set_max_drug_per_hour.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_panel_set_max_drug_per_hour :
 -----------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 141     | Inference          | Unchecked |   IU   |
| 2   | start |    assert @ finish  | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/
    panel_set_minimum_time_between_bolus.vcg
procedure Pca_Operation.Panel_Set_Minimum_Time_Between_Bolus

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:20

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/
    panel_set_minimum_time_between_bolus.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:1

VCs for procedure_panel_set_minimum_time_between_bolus :
 -----------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
```

```
| 1   | start | rtc check @ 153     | Inference          | Unchecked |   IU   |
| 2   | start |    assert @ finish  | Examiner           | Live      |   EL   |
 ---------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_set_vtbi.vcg
procedure Pca_Operation.Panel_Set_Vtbi

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:20

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/panel_set_vtbi.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:2

VCs for procedure_panel_set_vtbi :
 ---------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|---------------------------------------------------------------------------
| 1   | start | rtc check @ 129     | Inference          | Unchecked |   IU   |
| 2   | start |    assert @ finish  | Examiner           | Live      |   EL   |
 ---------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/patientbolus.vcg
procedure Pca_Operation.PatientBolus

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:20

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/patientbolus.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:2

VCs for procedure_patientbolus :
 ---------------------------------------------------------------------------
| #   | From  | To                  | Proved By          | Dead Path | Status |
|---------------------------------------------------------------------------
| 1   | start | rtc check @ 182     | Inference          | Unchecked |   IU   |
| 2   | start | rtc check @ 183     | Inference          | Unchecked |   IU   |
| 3   | start | rtc check @ 184     | Inference          | Unchecked |   IU   |
| 4   | start | rtc check @ 184     | Undischarged       | Unchecked |   UU   |
| 5   | start | rtc check @ 185     | Inference          | Unchecked |   IU   |
| 6   | start | rtc check @ 186     | Inference          | Unchecked |   IU   |
| 7   | start | rtc check @ 190     | Inference          | Unchecked |   IU   |
| 8   | start | rtc check @ 190     | Inference          | Unchecked |   IU   |
| 9   | start |    assert @ finish  | Examiner           | Live      |   EL   |
| 10  | start |    assert @ finish  | Examiner           | Live      |   EL   |
| 11  | start |    assert @ finish  | Examiner           | Live      |   EL   |
 ---------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/rate_controller.vcg
task_type Pca_Operation.Rate_Controller

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:28

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/rate_controller.dpc
DPCs generated 24-JUL-2014 11:39:12
```

DPC ZombieScoped 24-JUL-2014 11:39:2

VCs **for** task_type_rate_controller :

#	From	To	Proved By	Dead Path	Status
1	start	rtc check @ 231	Inference	Unchecked	IU
2	234	rtc check @ 231	Inference	Unchecked	IU
3	234	rtc check @ 231	Inference	Unchecked	IU
4	234	rtc check @ 231	Inference	Unchecked	IU
5	234	rtc check @ 231	Inference	Unchecked	IU
6	234	rtc check @ 231	Inference	Unchecked	IU
7	234	rtc check @ 231	Inference	Unchecked	IU
8	234	rtc check @ 231	Inference	Unchecked	IU
9	234	rtc check @ 231	Inference	Unchecked	IU
10	234	rtc check @ 231	Inference	Unchecked	IU
11	234	rtc check @ 231	Inference	Unchecked	IU
12	234	rtc check @ 231	Inference	Unchecked	IU
13	234	rtc check @ 231	Inference	Unchecked	IU
14	234	rtc check @ 231	Inference	Unchecked	IU
15	234	rtc check @ 231	Inference	Unchecked	IU
16	234	rtc check @ 231	Inference	Unchecked	IU
17	234	rtc check @ 231	Inference	Unchecked	IU
18	234	rtc check @ 231	Inference	Unchecked	IU
19	234	rtc check @ 231	Inference	Unchecked	IU
20	234	rtc check @ 231	Inference	Unchecked	IU
21	234	rtc check @ 231	Inference	Unchecked	IU
22	234	rtc check @ 231	Inference	Unchecked	IU
23	234	rtc check @ 231	Inference	Unchecked	IU
24	234	rtc check @ 231	Inference	Unchecked	IU
25	234	rtc check @ 231	Inference	Unchecked	IU
26	234	rtc check @ 231	Inference	Unchecked	IU
27	234	rtc check @ 231	Inference	Unchecked	IU
28	234	rtc check @ 231	Inference	Unchecked	IU
29	234	rtc check @ 231	Inference	Unchecked	IU
30	234	rtc check @ 231	Inference	Unchecked	IU
31	start	rtc check @ 232	Inference	Unchecked	IU
32	234	rtc check @ 232	Inference	Unchecked	IU
33	234	rtc check @ 232	Inference	Unchecked	IU
34	234	rtc check @ 232	Inference	Unchecked	IU
35	234	rtc check @ 232	Inference	Unchecked	IU
36	234	rtc check @ 232	Inference	Unchecked	IU
37	234	rtc check @ 232	Inference	Unchecked	IU
38	234	rtc check @ 232	Inference	Unchecked	IU
39	234	rtc check @ 232	Inference	Unchecked	IU
40	234	rtc check @ 232	Inference	Unchecked	IU
41	234	rtc check @ 232	Inference	Unchecked	IU
42	234	rtc check @ 232	Inference	Unchecked	IU
43	234	rtc check @ 232	Inference	Unchecked	IU
44	234	rtc check @ 232	Inference	Unchecked	IU
45	234	rtc check @ 232	Inference	Unchecked	IU
46	234	rtc check @ 232	Inference	Unchecked	IU
47	234	rtc check @ 232	Inference	Unchecked	IU
48	234	rtc check @ 232	Inference	Unchecked	IU
49	234	rtc check @ 232	Inference	Unchecked	IU
50	234	rtc check @ 232	Inference	Unchecked	IU
51	234	rtc check @ 232	Inference	Unchecked	IU
52	234	rtc check @ 232	Inference	Unchecked	IU
53	234	rtc check @ 232	Inference	Unchecked	IU
54	234	rtc check @ 232	Inference	Unchecked	IU
55	234	rtc check @ 232	Inference	Unchecked	IU
56	234	rtc check @ 232	Inference	Unchecked	IU
57	234	rtc check @ 232	Inference	Unchecked	IU
58	234	rtc check @ 232	Inference	Unchecked	IU

59	234	rtc check @ 232	Inference	Unchecked	IU
60	234	rtc check @ 232	Inference	Unchecked	IU
61	start	assert @ 234	Examiner	Live	EL
62	234	assert @ 234	Examiner	Live	EL
63	234	assert @ 234	Examiner	Live	EL
64	234	assert @ 234	Examiner	Live	EL
65	234	assert @ 234	Examiner	Live	EL
66	234	assert @ 234	Examiner	Live	EL
67	234	assert @ 234	Examiner	Live	EL
68	234	assert @ 234	Examiner	Live	EL
69	234	assert @ 234	Examiner	Live	EL
70	234	assert @ 234	Examiner	Live	EL
71	234	assert @ 234	Examiner	Live	EL
72	234	assert @ 234	Examiner	Live	EL
73	234	assert @ 234	Examiner	Live	EL
74	234	assert @ 234	Examiner	Live	EL
75	234	assert @ 234	Examiner	Live	EL
76	234	assert @ 234	Examiner	Live	EL
77	234	assert @ 234	Examiner	Live	EL
78	234	assert @ 234	Examiner	Live	EL
79	234	assert @ 234	Examiner	Live	EL
80	234	assert @ 234	Examiner	Live	EL
81	234	assert @ 234	Examiner	Live	EL
82	234	assert @ 234	Examiner	Live	EL
83	234	assert @ 234	Examiner	Live	EL
84	234	assert @ 234	Examiner	Live	EL
85	234	assert @ 234	Examiner	Live	EL
86	234	assert @ 234	Examiner	Live	EL
87	234	assert @ 234	Examiner	Live	EL
88	234	assert @ 234	Examiner	Live	EL
89	234	assert @ 234	Examiner	Live	EL
90	234	assert @ 234	Examiner	Live	EL
91	234	rtc check @ 240	Inference	Unchecked	IU
92	234	rtc check @ 240	Inference	Unchecked	IU
93	234	rtc check @ 241	Inference	Unchecked	IU
94	234	rtc check @ 241	Undischarged	Unchecked	UU
95	234	rtc check @ 242	Inference	Unchecked	IU
96	234	rtc check @ 243	Inference	Unchecked	IU
97	234	pre check @ 244	Inference	Unchecked	IU
98	234	rtc check @ 247	Inference	Unchecked	IU
99	234	rtc check @ 248	Inference	Unchecked	IU
100	234	rtc check @ 248	Inference	Unchecked	IU
101	234	rtc check @ 249	Inference	Unchecked	IU
102	234	rtc check @ 249	Undischarged	Unchecked	UU
103	234	rtc check @ 250	Inference	Unchecked	IU
104	234	rtc check @ 251	Inference	Unchecked	IU
105	234	pre check @ 252	Inference	Unchecked	IU
106	234	rtc check @ 256	Inference	Unchecked	IU
107	234	rtc check @ 257	Inference	Unchecked	IU
108	234	rtc check @ 257	Inference	Unchecked	IU
109	234	rtc check @ 258	Inference	Unchecked	IU
110	234	rtc check @ 258	Undischarged	Unchecked	UU
111	234	rtc check @ 259	Inference	Unchecked	IU
112	234	rtc check @ 260	Inference	Unchecked	IU
113	234	pre check @ 261	Inference	Unchecked	IU
114	234	rtc check @ 265	Inference	Unchecked	IU
115	234	rtc check @ 265	Inference	Unchecked	IU
116	234	rtc check @ 265	Inference	Unchecked	IU
117	234	rtc check @ 265	Inference	Unchecked	IU
118	234	rtc check @ 266	Inference	Unchecked	IU
119	234	rtc check @ 266	Inference	Unchecked	IU
120	234	rtc check @ 266	Undischarged	Unchecked	UU
121	234	rtc check @ 266	Undischarged	Unchecked	UU
122	234	rtc check @ 267	Inference	Unchecked	IU
123	234	rtc check @ 267	Inference	Unchecked	IU

124	234	rtc check @ 270	Inference	Unchecked	IU
125	234	rtc check @ 270	Inference	Unchecked	IU
126	234	rtc check @ 271	Undischarged	Unchecked	UU
127	234	rtc check @ 271	Undischarged	Unchecked	UU
128	234	rtc check @ 272	Inference	Unchecked	IU
129	234	rtc check @ 272	Inference	Unchecked	IU
130	234	rtc check @ 275	Inference	Unchecked	IU
131	234	rtc check @ 275	Inference	Unchecked	IU
132	234	rtc check @ 275	Inference	Unchecked	IU
133	234	rtc check @ 275	Inference	Unchecked	IU
134	234	pre check @ 276	Inference	Unchecked	IU
135	234	pre check @ 276	Inference	Unchecked	IU
136	234	pre check @ 276	Inference	Unchecked	IU
137	234	pre check @ 276	Inference	Unchecked	IU
138	234	rtc check @ 278	Undischarged	Unchecked	UU
139	234	rtc check @ 278	Undischarged	Unchecked	UU
140	234	rtc check @ 278	Undischarged	Unchecked	UU
141	234	rtc check @ 278	Undischarged	Unchecked	UU
142	234	rtc check @ 282	Inference	Unchecked	IU
143	234	rtc check @ 282	Inference	Unchecked	IU
144	234	rtc check @ 282	Inference	Unchecked	IU
145	234	rtc check @ 282	Inference	Unchecked	IU
146	234	rtc check @ 285	Inference	Unchecked	IU
147	234	rtc check @ 285	Inference	Unchecked	IU
148	234	rtc check @ 285	Inference	Unchecked	IU
149	234	rtc check @ 285	Inference	Unchecked	IU
150	234	rtc check @ 291	Inference	Unchecked	IU
151	234	rtc check @ 292	Inference	Unchecked	IU
152	234	rtc check @ 292	Inference	Unchecked	IU
153	234	rtc check @ 293	Inference	Unchecked	IU
154	234	rtc check @ 293	Undischarged	Unchecked	UU
155	234	rtc check @ 294	Inference	Unchecked	IU
156	234	rtc check @ 295	Inference	Unchecked	IU
157	234	pre check @ 296	Inference	Unchecked	IU
158	234	rtc check @ 300	Inference	Unchecked	IU
159	234	rtc check @ 300	Inference	Unchecked	IU
160	234	rtc check @ 301	Inference	Unchecked	IU
161	234	rtc check @ 301	Inference	Unchecked	IU
162	234	rtc check @ 302	Undischarged	Unchecked	UU
163	234	rtc check @ 302	Undischarged	Unchecked	UU
164	234	rtc check @ 302	Inference	Unchecked	IU
165	234	rtc check @ 302	Inference	Unchecked	IU
166	234	rtc check @ 303	Inference	Unchecked	IU
167	234	rtc check @ 303	Inference	Unchecked	IU
168	234	rtc check @ 303	Undischarged	Unchecked	UU
169	234	rtc check @ 303	Undischarged	Unchecked	UU
170	234	rtc check @ 304	Inference	Unchecked	IU
171	234	rtc check @ 304	Inference	Unchecked	IU
172	234	rtc check @ 307	Inference	Unchecked	IU
173	234	rtc check @ 307	Inference	Unchecked	IU
174	234	rtc check @ 308	Inference	Unchecked	IU
175	234	rtc check @ 308	Inference	Unchecked	IU
176	234	rtc check @ 311	Inference	Unchecked	IU
177	234	rtc check @ 311	Inference	Unchecked	IU
178	234	rtc check @ 311	Inference	Unchecked	IU
179	234	rtc check @ 311	Inference	Unchecked	IU
180	234	pre check @ 312	Inference	Unchecked	IU
181	234	pre check @ 312	Inference	Unchecked	IU
182	234	pre check @ 312	Inference	Unchecked	IU
183	234	pre check @ 312	Inference	Unchecked	IU
184	234	rtc check @ 314	Undischarged	Unchecked	UU
185	234	rtc check @ 314	Undischarged	Unchecked	UU
186	234	rtc check @ 314	Undischarged	Unchecked	UU
187	234	rtc check @ 314	Undischarged	Unchecked	UU
188	234	rtc check @ 316	Inference	Unchecked	IU

```
| 189 | 234   | rtc check @ 316     | Inference          | Unchecked |    IU    |
| 190 | 234   | rtc check @ 316     | Inference          | Unchecked |    IU    |
| 191 | 234   | rtc check @ 316     | Inference          | Unchecked |    IU    |
| 192 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 193 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 194 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 195 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 196 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 197 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 198 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 199 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 200 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 201 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 202 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 203 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 204 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 205 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 206 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 207 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 208 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 209 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 210 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 211 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 212 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 213 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 214 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 215 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 216 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 217 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 218 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 219 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
| 220 | 234   |    assert @ finish  | Examiner           | Dead      |    ED    |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/startpump.vcg
procedure Pca_Operation.StartPump

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:36

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/startpump.dpc
DPCs generated 24-JUL-2014 11:39:12

DPC ZombieScoped 24-JUL-2014  11:39:3

VCs for procedure_startpump :
 -----------------------------------------------------------------------------
| #    | From  | To                  | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1    | start | rtc check @ 166     | Inference          | Unchecked |    IU    |
| 2    | start |    assert @ finish  | Examiner           | Live      |    EL    |
 -----------------------------------------------------------------------------


File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/stoppump.vcg
procedure Pca_Operation.StopPump

VCs generated 24-JUL-2014 11:39:12

VCs simplified 24-JUL-2014 11:39:36

File /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/stoppump.dpc
DPCs generated 24-JUL-2014 11:39:12
```

```
DPC ZombieScoped 24-JUL-2014  11:39:3

VCs for procedure_stoppump :
 -----------------------------------------------------------------------------
| #   | From  | To                   | Proved By          | Dead Path | Status |
|-----------------------------------------------------------------------------
| 1   | start | rtc check @ 173      | Inference          | Unchecked |   IU   |
| 2   | start |    assert @ finish   | Examiner           | Live      |   EL   |
 -----------------------------------------------------------------------------



================================================================================
Summary:

The following subprograms have VCs proved false:

   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/start_pumping.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/stop_pumping.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_engine/write_signal.vcg

The following subprograms have undischarged VCs (excluding those proved false):

   2  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/get_time_between_activations.
     vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/get.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/inc.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/put.vcg
   2  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/integer_array_store/sum.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/max_drug_per_hour_watcher.vcg
   1  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/patientbolus.vcg
  20  /Users/jj/aadl-medical/pca-pump-beagleboard/pca_ravenscar/pca_operation/rate_controller.vcg

Proof strategies used by subprograms
-------------------------------------------------------------------------
Total subprograms with at least one VC proved by examiner:            15
Total subprograms with at least one VC proved by simplifier:          20
Total subprograms with at least one VC proved by contradiction:        0
Total subprograms with at least one VC proved with user proof rule:     0
Total subprograms with at least one VC proved by Victor:               0
Total subprograms with at least one VC proved by Riposte:              0
Total subprograms with at least one VC proved using checker:           0
Total subprograms with at least one VC discharged by review:           0

Maximum extent of strategies used for fully proved subprograms:
-------------------------------------------------------------------------
Total subprograms with proof completed by examiner:                    0
Total subprograms with proof completed by simplifier:                 14
Total subprograms with proof completed with user defined rules:         0
Total subprograms with proof completed by Victor:                      0
Total subprograms with proof completed by Riposte:                     0
Total subprograms with proof completed by checker:                     0
Total subprograms with VCs discharged by review:                       0

Overall subprogram summary:
-------------------------------------------------------------------------
Total subprograms fully proved:                                       14
Total subprograms with at least one undischarged VC:                   8  <<<
Total subprograms with at least one false VC:                          3  <<<
                                                                    -----
Total subprograms for which VCs have been generated:                  25


ZombieScope Summary:
-------------------------------------------------------------------------
```

```
Total subprograms for which DPCs have been generated:                25
Total number subprograms with dead paths found:                       3
Total number of dead paths found:                                    32


VC summary:
-----------------------------------------------------------------------
Note: (User) denotes where the Simplifier has proved VCs using one or
      more user-defined proof rules.

Total VCs by type:
------------------
                    Total   Examiner Simplifier False    Undisc.
Assert/Post            96         80         12     3          1
Precondition           12          0         12     0          0
Check stmnt.            0          0          0     0          0
Runtime check         187          0        159     0         28
Refinem. VCs           10         10          0     0          0
Inherit. VCs            0          0          0     0          0
================================================================
Totals:               305         90        183     3         29 <<<
%Totals:                         30%        60%    1%        10%

===================== End of Semantic Analysis Summary ========================
```

Listing B.1: *POGS report for PCA Pump prototype*

Appendix C

Simplified PCA pump AADL models

This appendix contains simplified AADL/BLESS models. They were created based on AADL/BLESS models of PCA pump, created by Brian Larson.

```
property set BLESS_Properties is
  with AADL_Project;

  Supported_Operators : list of aadlstring applies to ( data );
  Supported_Relations : list of aadlstring applies to ( data );
  Radix : AADL_Project::Size_Units applies to ( data );
end BLESS_Properties;
```

Listing C.1: *BLESS_Properties property set*

```
property set BLESS is
  Assertion : aadlstring applies to ( all );
  Typed : aadlstring applies to ( all );
  Invariant : aadlstring applies to ( all );
end BLESS;
```

Listing C.2: *BLESS property set*

```
property set PCA_Properties is
  with PCA_Types;

  Drug_Library_Size : constant aadlinteger => 500;
  Fault_Log_Size : constant aadlinteger => 150;
  Event_Log_Size : constant aadlinteger => 1500;
  KVO_Rate_Constant : constant aadlinteger => 1;
  KVO_Rate : constant aadlinteger => PCA_Properties::KVO_Rate_Constant;
  Max_Rate : constant aadlinteger => 10;
end PCA_Properties;
```

Listing C.3: *PCA_Properties property set*

```
package BLESS_Types public
with Base_Types, BLESS_Properties, Data_Model, Memory_Properties, BLESS;

data Integer extends Base_Types::Integer
  properties  --operators and relation symbols defined for Integer
    BLESS::Typed => "integer";
    BLESS_Properties::Supported_Operators => ("+", "*", "-", "/", "mod", "rem", "**");
    BLESS_Properties::Supported_Relations => ("=", "!=", "<", "<=", ">=", ">");
end Integer;

data Natural extends Base_Types::Natural
  properties  --operators and relation symbols defined for Natural
    BLESS::Typed => "natural";
    BLESS_Properties::Supported_Operators => ("+", "*", "-", "/", "mod", "rem", "**");
    BLESS_Properties::Supported_Relations => ("=", "!=", "<", "<=", ">=", ">");
end Natural;

data Real extends Base_Types::Float
  properties  --operators and relation symbols defined for Float
    BLESS::Typed => "real";
    BLESS_Properties::Supported_Operators => ("+", "*", "-", "/", "**");
    BLESS_Properties::Supported_Relations => ("=", "!=", "<", "<=", ">=", ">");
end Real;

data String extends Base_Types::String
  properties  --operators and relation symbols defined for String
    BLESS::Typed => "string";
    BLESS_Properties::Supported_Operators => ("+", "-"); --just concatenation
    BLESS_Properties::Supported_Relations => ("=", "!=", "<", "<=", ">=", ">");
end String;

data Fixed_Point
  properties  --operators and relation symbols defined for fixed-point arithmetic
    BLESS::Typed => "fixed";
    BLESS_Properties::Supported_Operators => ("+", "*", "-", "/", "**");
    BLESS_Properties::Supported_Relations => ("=", "!=", "<", "<=", ">=", ">");
    Data_Model::Data_Representation => Integer;
end Fixed_Point;

data Time extends Base_Types::Integer_64  --in milliseconds
  properties  --operators and relation symbols defined for Time
    --don't have a way to say that Time may be multiplied or divided by scalar
    --but not another Time
    BLESS::Typed => "integer";
    BLESS_Properties::Supported_Operators => ("+", "*", "-", "/");
    BLESS_Properties::Supported_Relations => ("=", "!=", "<", "<=", ">=", ">");
end Time;

end BLESS_Types;
```

Listing C.4: *`BLESS_Types` package*

```
package ICE_Types
public
with Data_Model;
with Base_Types;
  data Milliliter
  properties
    Data_Model::Data_Representation => Integer;
    Data_Model::Base_Type => (classifier (Base_Types::Unsigned_16)); --two bytes for 0-1000 ml
    Data_Model::Integer_Range => 0 .. 1000;
    Data_Model::Measurement_Unit => "ml";
  end Milliliter;

  data Milliliter_Per_Hour
  properties
    Data_Model::Data_Representation => Integer;
    Data_Model::Base_Type => (classifier (Base_Types::Unsigned_16)); --two bytes for 0-1000 ml/hr
    Data_Model::Integer_Range => 0 .. 1000;
    Data_Model::Measurement_Unit => "ml_per_hr";
  end Milliliter_Per_Hour;

  data Microliter_Per_Hour
  properties
    Data_Model::Data_Representation => Integer;
    Data_Model::Base_Type => (classifier (Base_Types::Unsigned_16)); --two bytes for 0-1000 ul/hr
    Data_Model::Integer_Range => 0 .. 1000;
    Data_Model::Measurement_Unit => "ul_per_hr";
  end Microliter_Per_Hour;

  data Minute
  properties
    Data_Model::Data_Representation => Integer;
    Data_Model::Base_Type => (classifier (Base_Types::Unsigned_16)); --two bytes for 0-1000 minutes
    Data_Model::Integer_Range => 0 .. 1000;
    Data_Model::Measurement_Unit => "min";
  end Minute;

  data Alarm_Signal --according to IEC 60601-1-8/FDIS AAA.201.8 ALARM SIGNAL inactivation states
  properties
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => ("On", "Alarm_Off", "Alarm_Paused", "Audio_Off", "Audio_Paused");
  end Alarm_Signal;

  data Percent
  properties
    Data_Model::Data_Representation => Integer;
    Data_Model::Base_Type => (classifier (Base_Types::Unsigned_8)); --one byte for 0-100 percent
    Data_Model::Integer_Range => 0 .. 100;
  end Percent;

  data Minute_Count extends Base_Types::Integer
  end Minute_Count;

  data Second_Count extends Base_Types::Integer
  end Second_Count;

end ICE_Types;
```

Listing C.5: *ICE_Types package*

```
package PCA_Types
public
  with Base_Types, Data_Model, PCA_Properties, ICE_Types, BLESS_Types, BLESS;

data Alarm_Type
  properties
    BLESS::Typed=>"enumeration (
          No_Alarm,
          Pump_Overheated,
          Defective_Battery,
          Low_Battery,
              POST_Failure,
          RAM_Failure,
          ROM_failure,
          CPU_Failure,
          Thread_Monitor_Failure,
          Air_In_Line,
          Upstream_Occlusion,
          Downstream_Occlusion,
          Empty_Reservoir,
          Basal_Overinfusion,
          Bolus_Overinfusion,
          Square_Bolus_Overinfusion)";
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => (
      "No_Alarm",
      "Pump_Overheated",
      "Defective_Battery",
      "Low_Battery",
        "POST_Failure",
        "RAM_Failure",
        "ROM_failure",
        "CPU_Failure",
        "Thread_Monitor_Failure",
        "Air_In_Line",
        "Upstream_Occlusion",
        "Downstream_Occlusion",
        "Empty_Reservoir",
      "Basal_Overinfusion",
      "Bolus_Overinfusion",
      "Square_Bolus_Overinfusion");
end Alarm_Type;

data Warning_Type
  properties
    BLESS::Typed=>
      "enumeration (No_Warning,
          Over_Max_Drug_Per_Hour,
          Soft_Limit,
          Low_Reservoir,
          Priming_Failure,
          Basal_Underinfusion,
          Bolus_Underinfusion,
          Square_Bolus_Underinfusion,
          Input_Needed,
          Long_Pause,
          Drug_Not_In_Library,
          Hard_Limit_Violated,
          Voltage_OOR)";
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => (
      "No_Warning",
      "Over_Max_Drug_Per_Hour",
        "Soft_Limit",
        "Low_Reservoir",
```

```
        "Priming_Failure",
      "Basal_Underinfusion",
      "Bolus_Underinfusion",
      "Square_Bolus_Underinfusion",
      "Input_Needed",
      "Long_Pause",
      "Drug_Not_In_Library",
      "Hard_Limit_Violated",
      "Voltage_OOR");
end Warning_Type;

data Status_Type
  properties
    BLESS::Typed=>"enumeration (Stopped,Bolus,Basal,KVO,Square_Bolus)";
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => ("Stopped","Bolus","Basal","KVO","Square_Bolus");
end Status_Type;

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;

data Drug_Volume  --volume of VTBI
  properties
    BLESS::Typed=>"integer";
    Data_Model::Base_Type => (classifier(Base_Types::Integer_16));
    Data_Model::Measurement_Unit => "ml";
end Drug_Volume;

data Drug_Weight  --string representing what drug, conectration, and volume is in the reservoir
  properties
    BLESS::Typed=>"integer";
    Data_Model::Base_Type => (classifier(Base_Types::Integer_16));
    Data_Model::Measurement_Unit => "mg";
end Drug_Weight;

data Drug_Concentration  --string representing what drug, conectration, and volume is in the reservoir
  properties
    BLESS::Typed=>"integer";
    Data_Model::Base_Type => (classifier(Base_Types::Integer));
    Data_Model::Measurement_Unit => "mg/l";
end Drug_Concentration;

data Drug_Record  --holds pharmacy data for a drug that may be used with the pump
  properties
    BLESS::Typed =>
    "record (
      Amount : PCA_Types::Drug_Weight;               --The weight of the drug dissolved in the diluent (mg)
      Concentration : PCA_Types::Drug_Concentration; --Drug concentration; as prescribed
      Vtbi_Lower_Soft : PCA_Types::Drug_Volume;      --Lower soft limit of drug volume to be infused
      Vtbi_Lower_Hard : PCA_Types::Drug_Volume;      --Lower hard limit of drug volume to be infused
      Vtbi_Typical : PCA_Types::Drug_Volume;         --Typical drug volume to be infused
      Vtbi_Upper_Soft : PCA_Types::Drug_Volume;      --Upper soft limit of drug volume to be infused
      Vtbi_Upper_Hard : PCA_Types::Drug_Volume;      --Upper hard limit of drug volume to be infused
      Basal_Rate_Lower_Soft : PCA_Types::Flow_Rate;  --Lower soft limit of basal drug dose rate
      Basal_Rate_Lower_Hard : PCA_Types::Flow_Rate;  --Lower hard limit of basal drug dose rate
      Basal_Rate_Typical : PCA_Types::Flow_Rate;     --Typical basal drug dose rate
      Basal_Rate_Upper_Soft : PCA_Types::Flow_Rate;  --Upper soft limit of basal drug dose rate
      Basal_Rate_Upper_Hard : PCA_Types::Flow_Rate;  --Upper hard limit of basal drug dose rate
      Bolus_Typical : PCA_Types::Drug_Volume;        --Typical Value of Bolus Volume
      Square_Bolus_rate_typical : PCA_Types::Flow_Rate;        --Typical duration of clinician commanded bolus
```

```
    )";
  Data_Model::Data_Representation => Struct;
  Data_Model::Element_Names =>
    ( "Amount",                  --The weight of the drug dissolved in the diluent (mg)
    "Concentration",           --Drug concentration; as prescribed
    "Vtbi_Lower_Soft",         --Lower soft limit of drug volume to be infused
    "Vtbi_Lower_Hard",         --Lower hard limit of drug volume to be infused
    "Vtbi_Typical",            --Typical drug volume to be infused
    "Vtbi_Upper_Soft",         --Upper soft limit of drug volume to be infused
    "Vtbi_Upper_Hard",         --Upper hard limit of drug volume to be infused
    "Basal_Rate_Lower_Soft",   --Lower soft limit of basal drug dose rate
    "Basal_Rate_Lower_Hard",   --Lower hard limit of basal drug dose rate
    "Basal_Rate_Typical",      --Typical basal drug dose rate
    "Basal_Rate_Upper_Soft",   --Upper soft limit of basal drug dose rate
    "Basal_Rate_Upper_Hard",   --Upper hard limit of basal drug dose rate
    "Bolus_Typical",           --Typical Value of Bolus Volume
    "Square_Bolus_Rate_Typical"       --Typical rate of clinician commanded bolus
    );
  Data_Model::Base_Type =>
    ( classifier(Drug_Weight),          --amount
      classifier(Drug_Concentration),  --concentration
      classifier(Drug_Volume),      --vtbi_lower_soft
      classifier(Drug_Volume),      --vtbi_lower_hard
      classifier(Drug_Volume),      --vtbi_typical
      classifier(Drug_Volume),      --vtbi_upper_soft
      classifier(Drug_Volume),      --vtbi_upper_hard
      classifier(Flow_Rate),            --basal_rate_lower_soft
      classifier(Flow_Rate),            --basal_rate_lower_hard
      classifier(Flow_Rate),            --basal_rate_typical
      classifier(Flow_Rate),            --basal_rate_upper_soft
      classifier(Flow_Rate),            --basal_rate_upper_hard
      classifier(Drug_Volume),      --bolus_typical
      classifier(Flow_Rate)     --ssquare_bolus_rate_typical
    );
end Drug_Record;


data Drug_Library  --holds drug records for all drugs approved by the hospital pharmacy
  properties
    BLESS::Typed => "array [PCA_Properties::Drug_Library_Size] of PCA_Types::Drug_Record";
    Data_Model::Data_Representation => Array;
    Data_Model::Base_Type => (classifier(Drug_Record));
    Data_Model::Dimension => (PCA_Properties::Drug_Library_Size);
end Drug_Library;

data Prescription
  properties
    BLESS::Typed =>
    "record (
    Concentration : Drug_Concentration;
    Initial_Volume : Drug_Volume;
    Basal_Flow_Rate : Flow_Rate;
    Vtbi : Drug_Volume;
    Max_Drug_Per_Hour : Drug_Volume;
    Minimum_Time_Between_Bolus : ICE_Types::Minute;
    )";
    Data_Model::Data_Representation => Struct;
    Data_Model::Element_Names =>
      ( "Concentration",
    "Initial_Volume",
    "Basal_Flow_Rate",
    "Vtbi",
    "Max_Drug_Per_Hour",
    "Minimum_Time_Between_Bolus"
      );
```

```
    Data_Model::Base_Type =>
      ( classifier(Drug_Concentration),   --concentration
        classifier(Drug_Volume),          --initial volume
        classifier(Flow_Rate),            --basal flow rate
        classifier(Drug_Volume),          --VTBI
        classifier(Drug_Volume),          --maximum drug allowed per hour
        classifier(ICE_Types::Minute)    --min time between bolus doses
      );
end Prescription;

data Fault_Record  --record of fault for log
  properties
    BLESS::Typed => "record (Alarm:Alarm_Type; Warning:Warning_Type; Occurrence_Time:BLESS_Types::Time;)";
    Data_Model::Data_Representation => Struct;
    Data_Model::Element_Names => ("Alarm","Warning","Occurrence_Time");
    Data_Model::Base_Type => ( classifier(Alarm_Type),classifier(Warning_Type),classifier(BLESS_Types::Time))
     ;
end Fault_Record;

data Fault_Log  --holds records of faults
  properties
    BLESS::Typed => "array [PCA_Properties::Fault_Log_Size] of PCA_Types::Fault_Record";
    Data_Model::Data_Representation => Array;
    Data_Model::Base_Type => (classifier(Fault_Record));
    Data_Model::Dimension => (PCA_Properties::Fault_Log_Size);
end Fault_Log;

data Event_Record  --record of event for log
  properties
    BLESS::Typed => "record ( Time : BLESS_Types::Time)";
    Data_Model::Data_Representation => Struct;
    Data_Model::Element_Names => ( "Time" );
    Data_Model::Base_Type => (classifier(BLESS_Types::Time));
end Event_Record;

data Event_Log  --holds records of events
  properties
    BLESS::Typed => "array [PCA_Properties::Event_Log_Size] of PCA_Types::Event_Record";
    Data_Model::Data_Representation => Array;
    Data_Model::Base_Type => (classifier(Event_Record));
    Data_Model::Dimension => (PCA_Properties::Event_Log_Size);
end Event_Log;

data Infusion_Type   --used for over- and under-infusion alarms
  properties
    BLESS::Typed=>"enumeration(Bolus_Infusion, Square_Infusion, Basal_Infusion, KVO_Infusion)";
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => ("Bolus_Infusion","Square_Infusion","Basal_Infusion","KVO_Infusion");
end Infusion_Type;

data Pump_Fault_Type
  properties
    BLESS::Typed=>"enumeration(Prime_Failure, Pump_Hot, Bubble, Upstream_Occlusion_Fault,
     Downstream_Occlusion_Fault, Overinfusion, Underinfusion)";
    Data_Model::Data_Representation => Enum;
    Data_Model::Enumerators => ("Prime_Failure","Pump_Hot","Bubble","Upstream_Occlusion_Fault","
     Downstream_Occlusion_Fault","Overinfusion","Underinfusion" );
end Pump_Fault_Type;

end PCA_Types;
```

Listing C.6: `PCA_Types` *package*

```
package PCA_Operation
  public
  with PCA_Properties, Base_Types, BLESS, BLESS_Types, ICE_Types, PCA_Types;

system operation
  features
    Start_Button_Pressed: in event port;
    Stop_Button_Pressed: in event port;
    Patient_Request_Bolus: in event port;
    Clinician_Request_Bolus: in event port;
    Bolus_Duration: in event data port ICE_Types::Minute;
    Infusion_Flow_Rate: out data port PCA_Types::Flow_Rate;
    System_Status: out data port PCA_Types::Status_Type;
    Rx: in event data port PCA_Types::Prescription;
end operation;

system implementation operation.imp
  subcomponents
    operation_process: process operation_process.imp;
  connections
    start: port Start_Button_Pressed -> operation_process.Start_Button_Pressed;
    stop: port Stop_Button_Pressed -> operation_process.Stop_Button_Pressed;
    pbp: port Patient_Request_Bolus -> operation_process.Patient_Request_Bolus;
    crb: port Clinician_Request_Bolus -> operation_process.Clinician_Request_Bolus;
    bd: port Bolus_Duration -> operation_process.Bolus_Duration;
    pfr: port operation_process.Infusion_Flow_Rate -> Infusion_Flow_Rate;
    stat: port operation_process.System_Status -> System_Status;
    rxo: port Rx->operation_process.Rx;
end operation.imp;

process operation_process
  features
    Start_Button_Pressed: in event port;
    Stop_Button_Pressed: in event port;
    Patient_Request_Bolus: in event port;
    Clinician_Request_Bolus: in event port;
    Bolus_Duration: in event data port ICE_Types::Minute;
    Infusion_Flow_Rate: out data port PCA_Types::Flow_Rate;
    System_Status: out data port PCA_Types::Status_Type;
    Rx: in event data port PCA_Types::Prescription;
end operation_process;

process implementation operation_process.imp
  subcomponents
    Max_Drug_Per_Hour_Watcher : thread Max_Drug_Per_Hour_Watcher.imp;
    Rate_Controller : thread Rate_Controller.imp;
    Patient_Bolus_Checker : thread Patient_Bolus_Checker.imp;
  connections
    start: port Start_Button_Pressed -> Rate_Controller.Start_Button_Pressed;
    stop: port Stop_Button_Pressed -> Rate_Controller.Stop_Button_Pressed;
    pb: port Patient_Request_Bolus -> Patient_Bolus_Checker.Patient_Request_Bolus;
    crb: port Clinician_Request_Bolus -> Rate_Controller.Clinician_Request_Bolus;
    bd: port Bolus_Duration -> Rate_Controller.Bolus_Duration;
    pfr: port Rate_Controller.Infusion_Flow_Rate -> Infusion_Flow_Rate;
    ss: port Rate_Controller.System_Status -> System_Status;
    rxrc: port Rx->Rate_Controller.Rx;
end operation_process.imp;

thread Max_Drug_Per_Hour_Watcher
features
    Infusion_Flow_Rate: in data port PCA_Types::Flow_Rate
      {BLESS::Assertion => "<<:=PUMP_RATE()>>";};
    Max_Drug_Per_Hour: in data port PCA_Types::Drug_Volume
      {BLESS::Assertion => "<<:=MAX_DRUG_PER_HOUR>>";};
end Max_Drug_Per_Hour_Watcher;
```

```

thread implementation Max_Drug_Per_Hour_Watcher.imp
end Max_Drug_Per_Hour_Watcher.imp;

thread Rate_Controller
features
  Start_Button_Pressed: in event port;
  Stop_Button_Pressed: in event port;
  Rx: in event data port PCA_Types::Prescription
    {BLESS::Assertion => "<<:=Rx_APPROVED()>>";};
    Clinician_Request_Bolus: in event port;
    Bolus_Duration: in event data port ICE_Types::Minute;
    Infusion_Flow_Rate: out data port PCA_Types::Flow_Rate
      {BLESS::Assertion => "<<:=PUMP_RATE()>>";};
    System_Status: out event data port PCA_Types::Status_Type;
end Rate_Controller;

thread implementation Rate_Controller.imp
annex BLESS
{**
assert
<<HALT : :(la=StopButton) >>                              --pump at 0 if stop button
<<KVO_RATE : :(la=TooMuchJuice)>>                           --pump at KVO rate when commanded, some alarms, or exceeded hourly limit
<<PB_RATE : :la=PatientButton>>                           --patient button pressed, and allowed
<<CCB_RATE : :(la=StartSquareBolus) or (la=ResumeSquareBolus)>>         --clinician-commanded bolus start or resumption after patient bolus
<<BASAL_RATE : :(la=StartButton) or (la=ResumeBasal) or (la=SquareBolusDone)>>  --regular infusion
<<PUMP_RATE : :=
  (HALT()) -> 0,                                        --no flow
  (KVO_RATE()) -> PCA_Properties::KVO_Rate,             --KVO rate
  (PB_RATE()) -> Patient_Bolus_Rate,              --maximum infusion upon patient request
  (CCB_RATE()) -> Square_Bolus_Rate,                    --square bolus rate=VTBI/duration, from data port
  (BASAL_RATE()) -> Basal_Rate                          --basal rate, from data port
>>
invariant <<true>>
variables
  la :   --last action
    enumeration (
      StopButton,       --clinician pressed stop button
      TooMuchJuice,         --exceeded max drug per hour, pump at KVO until prescription and patient are re-authenticated
      PatientButton,      --patient requested drug
      ResumeSquareBolus,    --infusion of VTBI finished, resume clinician-commanded bolus
      ResumeBasal,        --infusion of VTBI finished, resume basal-rate
      StartSquareBolus,     --begin clinician-commanded bolus
      SquareBolusDone,      --infusion of VTBI finished
      StartButton       --start pumping at basal rate
    );
**};
end Rate_Controller.imp;

thread Patient_Bolus_Checker
  features
    Patient_Request_Bolus: in event port;
end Patient_Bolus_Checker;

thread implementation Patient_Bolus_Checker.imp
end Patient_Bolus_Checker.imp;

end PCA_Operation;
```

Listing C.7: `PCA_Operation` *package*

Appendix D

PCA pump Prototype - translated from simplified AADL models

This appendinx presents PCA pump prototype, which was created by direct translation from simplified AADL/BLESS models presented in appendix C.

```
package Base_Types
is
    protected type Boolean_Store
    is
        pragma Priority (10);

        function Get return Boolean;
        --# global in Boolean_Store;

        procedure Put(X : in Boolean);
        --# global out Boolean_Store;
        --# derives Boolean_Store from X;
    private
        TheStoredData : Boolean := False;
    end Boolean_Store;

    protected type Integer_Store
    is
        pragma Priority (10);

        function Get return Integer;
        --# global in Integer_Store;

        procedure Put(X : in Integer);
        --# global out Integer_Store;
        --# derives Integer_Store from X;
    private
        TheStoredData : Integer := 0;
    end Integer_Store;
```

```
protected type Natural_Store
is
    pragma Priority (10);

    function Get return Natural;
    --# global in Natural_Store;

    procedure Put(X : in Natural);
    --# global out Natural_Store;
    --# derives Natural_Store from X;
private
    TheStoredData : Natural := 0;
end Natural_Store;

type Integer_8 is new Integer range -2**(1*8-1) .. 2**(1*8-1)-1;

protected type Integer_8_Store
is
    pragma Priority (10);

    function Get return Integer_8;
    --# global in Integer_8_Store;

    procedure Put(X : in Integer_8);
    --# global out Integer_8_Store;
    --# derives Integer_8_Store from X;
private
    TheStoredData : Integer_8 := 0;
end Integer_8_Store;

type Integer_16 is new Integer range -2**(2*8-1) .. 2**(2*8-1)-1;

protected type Integer_16_Store
is
    pragma Priority (10);

    function Get return Integer_16;
    --# global in Integer_16_Store;

    procedure Put(X : in Integer_16);
    --# global out Integer_16_Store;
    --# derives Integer_16_Store from X;
private
    TheStoredData : Integer_16 := 0;
end Integer_16_Store;

type Integer_32 is new Integer range -2**(4*8-1) .. 2**(4*8-1)-1;

protected type Integer_32_Store
is
    pragma Priority (10);

    function Get return Integer_32;
    --# global in Integer_32_Store;

    procedure Put(X : in Integer_32);
    --# global out Integer_32_Store;
    --# derives Integer_32_Store from X;
private
    TheStoredData : Integer_32 := 0;
end Integer_32_Store;

type Integer_64 is range -2**(8*8-1) .. 2**(8*8-1)-1; -- with new Integer gnat compiler error: value not
 in range of type "Standard.Integer"
```

```
protected type Integer_64_Store
is
    pragma Priority (10);

    function Get return Integer_64;
    --# global in Integer_64_Store;

    procedure Put(X : in Integer_64);
    --# global out Integer_64_Store;
    --# derives Integer_64_Store from X;
private
    TheStoredData : Integer_64 := 0;
end Integer_64_Store;

type Unsigned_8 is new Integer range 0 .. 2**(1*8)-1;

protected type Unsigned_8_Store
is
    pragma Priority (10);

    function Get return Unsigned_8;
    --# global in Unsigned_8_Store;

    procedure Put(X : in Unsigned_8);
    --# global out Unsigned_8_Store;
    --# derives Unsigned_8_Store from X;
private
    TheStoredData : Unsigned_8 := 0;
end Unsigned_8_Store;

type Unsigned_16 is new Integer range 0 .. 2**(2*8)-1;

protected type Unsigned_16_Store
is
    pragma Priority (10);

    function Get return Unsigned_16;
    --# global in Unsigned_16_Store;

    procedure Put(X : in Unsigned_16);
    --# global out Unsigned_16_Store;
    --# derives Unsigned_16_Store from X;
private
    TheStoredData : Unsigned_16 := 0;
end Unsigned_16_Store;

type Unsigned_32 is range 0 .. 2**(4*8)-1; -- with new Integer gnat compiler error: value not in range of type "Standard.Integer"

protected type Unsigned_32_Store
is
    pragma Priority (10);

    function Get return Unsigned_32;
    --# global in Unsigned_32_Store;

    procedure Put(X : in Unsigned_32);
    --# global out Unsigned_32_Store;
    --# derives Unsigned_32_Store from X;
private
    TheStoredData : Unsigned_32 := 0;
end Unsigned_32_Store;
```

```
    --type Unsigned_64 is range 0 .. 2**64-1; -- gnat compiler error: integer type definition bounds out of
     range

end Base_Types;

package body Base_Types
is
    protected body Boolean_Store is
        function Get return Boolean
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Boolean)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Boolean_Store;

    protected body Integer_Store is
        function Get return Integer
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Integer)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Integer_Store;

    protected body Natural_Store is
        function Get return Natural
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Natural)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Natural_Store;

    protected body Integer_8_Store is
        function Get return Integer_8
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;
```

```

        procedure Put(X : in Integer_8)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Integer_8_Store;

    protected body Integer_16_Store is
        function Get return Integer_16
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Integer_16)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Integer_16_Store;

    protected body Integer_32_Store is
        function Get return Integer_32
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Integer_32)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Integer_32_Store;

    protected body Integer_64_Store is
        function Get return Integer_64
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Integer_64)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Integer_64_Store;

    protected body Unsigned_8_Store is
        function Get return Unsigned_8
        --# global in TheStoredData;
        is
```

```
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Unsigned_8)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Unsigned_8_Store;

    protected body Unsigned_16_Store is
        function Get return Unsigned_16
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Unsigned_16)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Unsigned_16_Store;

    protected body Unsigned_32_Store is
        function Get return Unsigned_32
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Unsigned_32)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Unsigned_32_Store;

end Base_Types;
```

Listing D.1: `Base_Types` *package*

```
with Base_Types;
--# inherit Base_Types;
package Bless_Types
is
    subtype Fixed_Point is Integer;
    protected type Fixed_Point_Store
    is
        pragma Priority (10);
        function Get return Fixed_Point;
        --# global in Fixed_Point_Store;
        procedure Put(X : in Fixed_Point);
        --# global out Fixed_Point_Store;
        --# derives Fixed_Point_Store from X;
    private
        TheStoredData : Fixed_Point := 0;
    end Fixed_Point_Store;

    subtype Time is Base_Types.Integer_64;
    protected type Time_Store
    is
        pragma Priority (10);
        function Get return Time;
        --# global in Time_Store;
        procedure Put(X : in Time);
        --# global out Time_Store;
        --# derives Time_Store from X;
    private
        TheStoredData : Time := 0;
    end Time_Store;
end Bless_Types;

package body Bless_Types
is
    protected body Fixed_Point_Store is
        function Get return Fixed_Point
        --# global in TheStoredData;
        is begin
            return TheStoredData;
        end Get;
        procedure Put(X : in Fixed_Point)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is begin
            TheStoredData := X;
        end Put;
    end Fixed_Point_Store;

    protected body Time_Store is
        function Get return Time
        --# global in TheStoredData;
        is begin
            return TheStoredData;
        end Get;
        procedure Put(X : in Time)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is begin
            TheStoredData := X;
        end Put;
    end Time_Store;
end Bless_Types;
```

Listing D.2: `Bless_Types` *package*

```
with Base_Types;
--# inherit Base_Types;
package Ice_Types
is
    subtype Milliliter is Base_Types.Unsigned_16 range 0 .. 1000;

    protected type Milliliter_Store
    is
        pragma Priority (10);

        function Get return Milliliter;
        --# global in Milliliter_Store;

        procedure Put(X : in Milliliter);
        --# global out Milliliter_Store;
        --# derives Milliliter_Store from X;
    private
        TheStoredData : Milliliter := 0;
    end Milliliter_Store;


    subtype Milliliter_Per_Hour is Base_Types.Unsigned_16 range 0 .. 1000;

    protected type Milliliter_Per_Hour_Store
    is
        pragma Priority (10);

        function Get return Milliliter_Per_Hour;
        --# global in Milliliter_Per_Hour_Store;

        procedure Put(X : in Milliliter_Per_Hour);
        --# global out Milliliter_Per_Hour_Store;
        --# derives Milliliter_Per_Hour_Store from X;
    private
        TheStoredData : Milliliter_Per_Hour := 0;
    end Milliliter_Per_Hour_Store;


    subtype Microliter_Per_Hour is Base_Types.Unsigned_16 range 0 .. 1000;

    protected type Microliter_Per_Hour_Store
    is
        pragma Priority (10);

        function Get return Microliter_Per_Hour;
        --# global in Microliter_Per_Hour_Store;

        procedure Put(X : in Microliter_Per_Hour);
        --# global out Microliter_Per_Hour_Store;
        --# derives Microliter_Per_Hour_Store from X;
    private
        TheStoredData : Microliter_Per_Hour := 0;
    end Microliter_Per_Hour_Store;


    subtype Minute is Base_Types.Unsigned_16 range 0 .. 1000;

    protected type Minute_Store
    is
        pragma Priority (10);

        function Get return Minute;
        --# global in Minute_Store;

        procedure Put(X : in Minute);
```

```
    --# global out Minute_Store;
    --# derives Minute_Store from X;
private
    TheStoredData : Minute := 0;
end Minute_Store;


type Alarm_Signal is (On, Alarm_Off, Alarm_Paused, Audio_Off, Audio_Paused);

protected type Alarm_Signal_Store
is
    pragma Priority (10);

    function Get return Alarm_Signal;
    --# global in Alarm_Signal_Store;

    procedure Put(X : in Alarm_Signal);
    --# global out Alarm_Signal_Store;
    --# derives Alarm_Signal_Store from X;
private
    TheStoredData : Alarm_Signal := Alarm_Signal'First;
end Alarm_Signal_Store;


subtype Percent is Base_Types.Unsigned_8 range 0 .. 100;

protected type Percent_Store
is
    pragma Priority (10);

    function Get return Percent;
    --# global in Percent_Store;

    procedure Put(X : in Percent);
    --# global out Percent_Store;
    --# derives Percent_Store from X;
private
    TheStoredData : Percent := 0;
end Percent_Store;


type Minute_Count is new Integer;

protected type Minute_Count_Store
is
    pragma Priority (10);

    function Get return Minute_Count;
    --# global in Minute_Count_Store;

    procedure Put(X : in Minute_Count);
    --# global out Minute_Count_Store;
    --# derives Minute_Count_Store from X;
private
    TheStoredData : Minute_Count := 0;
end Minute_Count_Store;


type Second_Count is new Integer;

protected type Second_Count_Store
is
    pragma Priority (10);

    function Get return Second_Count;
```

```
        --# global in Second_Count_Store;

        procedure Put(X : in Second_Count);
        --# global out Second_Count_Store;
        --# derives Second_Count_Store from X;
    private
        TheStoredData : Second_Count := 0;
    end Second_Count_Store;
end Ice_Types;

package body Ice_Types
is
    protected body Milliliter_Store is
        function Get return Milliliter
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Milliliter)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Milliliter_Store;

    protected body Milliliter_Per_Hour_Store is
        function Get return Milliliter_Per_Hour
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Milliliter_Per_Hour)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Milliliter_Per_Hour_Store;

    protected body Microliter_Per_Hour_Store is
        function Get return Microliter_Per_Hour
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Microliter_Per_Hour)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Microliter_Per_Hour_Store;

    protected body Minute_Store is
        function Get return Minute
```

```
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Minute)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Minute_Store;

    protected body Alarm_Signal_Store is
        function Get return Alarm_Signal
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Alarm_Signal)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Alarm_Signal_Store;

    protected body Percent_Store is
        function Get return Percent
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Percent)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Percent_Store;

    protected body Minute_Count_Store is
        function Get return Minute_Count
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Minute_Count)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Minute_Count_Store;
```

```

    protected body Second_Count_Store is
        function Get return Second_Count
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Second_Count)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Second_Count_Store;

end Ice_Types;
```

Listing D.3: *Ice_Types package*

```
with Base_Types;
with Bless_Types;
with Ice_Types;
with Pca_Properties;
--# inherit Base_Types,
--#         Bless_Types,
--#         Ice_Types,
--#         Pca_Properties;
package Pca_Types
is
    type Alarm_Type is (
                        No_Alarm,
                        Pump_Overheated,
                        Defective_Battery,
                        Low_Battery,
                        POST_Failure,
                        RAM_Failure,
                        ROM_failure,
                        CPU_Failure,
                        Thread_Monitor_Failure,
                        Air_In_Line,
                        Upstream_Occlusion,
                        Downstream_Occlusion,
                        Empty_Reservoir,
                        Basal_Overinfusion,
                        Bolus_Overinfusion,
                        Square_Bolus_Overinfusion
                        );

    protected type Alarm_Type_Store
    is
        pragma Priority (10);

        function Get return Alarm_Type;
        --# global in Alarm_Type_Store;

        procedure Put(X : in Alarm_Type);
        --# global out Alarm_Type_Store;
        --# derives Alarm_Type_Store from X;
    private
        TheStoredData : Alarm_Type := Alarm_Type'First;
```

```
    end Alarm_Type_Store;


    type Warning_Type is (No_Warning,
                          Over_Max_Drug_Per_Hour,
                          Soft_Limit,
                          Low_Reservoir,
                          Priming_Failure,
                          Basal_Underinfusion,
                          Bolus_Underinfusion,
                          Square_Bolus_Underinfusion,
                          Input_Needed,
                          Long_Pause,
                          Drug_Not_In_Library,
                          Hard_Limit_Violated,
                          Voltage_OOR
                          );

    protected type Warning_Type_Store
    is
        pragma Priority (10);

        function Get return Warning_Type;
        --# global in Warning_Type_Store;

        procedure Put(X : in Warning_Type);
        --# global out Warning_Type_Store;
        --# derives Warning_Type_Store from X;
    private
        TheStoredData : Warning_Type := Warning_Type'First;
    end Warning_Type_Store;


    type Status_Type is (Stopped, Bolus, Basal, KVO, Square_Bolus);

    protected type Status_Type_Store
    is
        pragma Priority (10);

        function Get return Status_Type;
        --# global in Status_Type_Store;

        procedure Put(X : in Status_Type);
        --# global out Status_Type_Store;
        --# derives Status_Type_Store from X;
    private
        TheStoredData : Status_Type := Status_Type'First;
    end Status_Type_Store;


    subtype Flow_Rate is Base_Types.Integer_16;

    protected type Flow_Rate_Store
    is
        pragma Priority (10);

        function Get return Flow_Rate;
        --# global in Flow_Rate_Store;

        procedure Put(X : in Flow_Rate);
        --# global out Flow_Rate_Store;
        --# derives Flow_Rate_Store from X;
    private
        TheStoredData : Flow_Rate := 0;
    end Flow_Rate_Store;
```

```
subtype Drug_Volume is Base_Types.Integer_16;

protected type Drug_Volume_Store
is
    pragma Priority (10);

    function Get return Drug_Volume;
    --# global in Drug_Volume_Store;

    procedure Put(X : in Drug_Volume);
    --# global out Drug_Volume_Store;
    --# derives Drug_Volume_Store from X;
private
    TheStoredData : Drug_Volume := 0;
end Drug_Volume_Store;


subtype Drug_Weight is Base_Types.Integer_16;

protected type Drug_Weight_Store
is
    pragma Priority (10);

    function Get return Drug_Weight;
    --# global in Drug_Weight_Store;

    procedure Put(X : in Drug_Weight);
    --# global out Drug_Weight_Store;
    --# derives Drug_Weight_Store from X;
private
    TheStoredData : Drug_Weight := 0;
end Drug_Weight_Store;


type Drug_Concentration is new Integer;

protected type Drug_Concentration_Store
is
    pragma Priority (10);

    function Get return Drug_Concentration;
    --# global in Drug_Concentration_Store;

    procedure Put(X : in Drug_Concentration);
    --# global out Drug_Concentration_Store;
    --# derives Drug_Concentration_Store from X;
private
    TheStoredData : Drug_Concentration := 0;
end Drug_Concentration_Store;


type Drug_Record is record
    Amount : Drug_Weight;
    Concentration : Drug_Concentration;
    Vtbi_Lower_Soft : Drug_Volume;
    Vtbi_Lower_Hard : Drug_Volume;
    Vtbi_Typical : Drug_Volume;
    Vtbi_Upper_Soft : Drug_Volume;
    Vtbi_Upper_Hard : Drug_Volume;
    Basal_Rate_Lower_Soft : Flow_Rate;
    Basal_Rate_Lower_Hard : Flow_Rate;
    Basal_Rate_Typical : Flow_Rate;
    Basal_Rate_Upper_Soft : Flow_Rate;
```

```
      Basal_Rate_Upper_Hard : Flow_Rate;
      Bolus_Typical : Drug_Volume;
      Bolus_Time_Typical : Ice_Types.Minute;
   end record;

   protected type Drug_Record_Store
   is
      pragma Priority (10);

      function Get return Drug_Record;
      --# global in Drug_Record_Store;

      procedure Put(X : in Drug_Record);
      --# global out Drug_Record_Store;
      --# derives Drug_Record_Store from X;
   private
      TheStoredData : Drug_Record :=
        Drug_Record'(Amount => Drug_Weight'First,
                     Concentration => Drug_Concentration'First,
                     Vtbi_Lower_Soft => Drug_Volume'First,
                     Vtbi_Lower_Hard => Drug_Volume'First,
                     Vtbi_Typical => Drug_Volume'First,
                     Vtbi_Upper_Soft => Drug_Volume'First,
                     Vtbi_Upper_Hard => Drug_Volume'First,
                     Basal_Rate_Lower_Soft => Flow_Rate'First,
                     Basal_Rate_Lower_Hard => Flow_Rate'First,
                     Basal_Rate_Typical => Flow_Rate'First,
                     Basal_Rate_Upper_Soft => Flow_Rate'First,
                     Basal_Rate_Upper_Hard => Flow_Rate'First,
                     Bolus_Typical => Drug_Volume'First,
                     Bolus_Time_Typical => Ice_Types.Minute'First
                     );
   end Drug_Record_Store;


   subtype Drug_Library_Index is Integer range 1 .. Pca_Properties.Drug_Library_Size;
   type Drug_Library is array (Drug_Library_Index) of Drug_Record;

   protected type Drug_Library_Store
   is
      pragma Priority (10);

      function Get(Ind : in Integer) return Drug_Record;
      --# global in Drug_Library_Store;

      procedure Put(Ind : in Integer; Val : in Drug_Record);
      --# global in out Drug_Library_Store;
      --# derives Drug_Library_Store from Drug_Library_Store, Ind, Val;
  private
     TheStoredData : Drug_Library := Drug_Library'(others =>
                         Drug_Record'(Amount => Drug_Weight'First,
                                      Concentration => Drug_Concentration'First,
                                      Vtbi_Lower_Soft => Drug_Volume'First,
                                      Vtbi_Lower_Hard => Drug_Volume'First,
                                      Vtbi_Typical => Drug_Volume'First,
                                      Vtbi_Upper_Soft => Drug_Volume'First,
                                      Vtbi_Upper_Hard => Drug_Volume'First,
                                      Basal_Rate_Lower_Soft => Flow_Rate'First,
                                      Basal_Rate_Lower_Hard => Flow_Rate'First,
                                      Basal_Rate_Typical => Flow_Rate'First,
                                      Basal_Rate_Upper_Soft => Flow_Rate'First,
                                      Basal_Rate_Upper_Hard => Flow_Rate'First,
                                      Bolus_Typical => Drug_Volume'First,
                                      Bolus_Time_Typical => Ice_Types.Minute'First
                                     ));
```

```
end Drug_Library_Store;

type Prescription is record
    Concentration : Drug_Concentration;
    Initial_Volume : Drug_Volume;
    Basal_Flow_Rate : Flow_Rate;
    Vtbi : Drug_Volume;
    Max_Drug_Per_Hour : Drug_Volume;
    Minimum_Time_Between_Bolus : Ice_Types.Minute;
end record;

protected type Prescription_Store
is
    pragma Priority (10);

    function Get return Prescription;
    --# global in Prescription_Store;

    procedure Put(Prescription_In : in Prescription);
    --# global out Prescription_Store;
    --# derives Prescription_Store from Prescription_In;

private
    TheStoredData : Prescription :=
      Prescription'(Concentration => 0,
                    Initial_Volume => 0,
                    Basal_Flow_Rate => 0,
                    Vtbi => 0,
                    Max_Drug_Per_Hour => 0,
                    Minimum_Time_Between_Bolus => 0
                   );

end Prescription_Store;

type Fault_Record is record
    Alarm : Alarm_Type;
    Warning : Warning_Type;
    Time : Bless_Types.Time;
end record;

protected type Fault_Record_Store
is
    pragma Priority (10);

    function Get return Fault_Record;
    --# global in Fault_Record_Store;

    procedure Put(X : in Fault_Record);
    --# global out Fault_Record_Store;
    --# derives Fault_Record_Store from X;
private
    TheStoredData : Fault_Record := Fault_Record'(Alarm => Alarm_Type'First,
                                                   Warning => Warning_Type'First,
                                                   Time => Bless_Types.Time'First
                                                  );
end Fault_Record_Store;


subtype Fault_Log_Index is Integer range 1 .. Pca_Properties.Fault_Log_Size;
type Fault_Log is array (Fault_Log_Index) of Fault_Record;

protected type Fault_Log_Store
is
    pragma Priority (10);

```

```
    function Get(Ind : in Integer) return Fault_Record;
    --# global in Fault_Log_Store;

    procedure Put(Ind : in Integer; Val : in Fault_Record);
    --# global in out Fault_Log_Store;
    --# derives Fault_Log_Store from Fault_Log_Store, Ind, Val;
private
    TheStoredData : Fault_Log := Fault_Log'(others =>
                                            Fault_Record'(Alarm => Alarm_Type'First,
                                                          Warning => Warning_Type'First,
                                                          Time => Bless_Types.Time'First
                                                         ));
end Fault_Log_Store;


type Event_Record is record
    Time : Bless_Types.Time;
end record;

protected type Event_Record_Store
is
    pragma Priority (10);

    function Get return Event_Record;
    --# global in Event_Record_Store;

    procedure Put(X : in Event_Record);
    --# global out Event_Record_Store;
    --# derives Event_Record_Store from X;
private
    TheStoredData : Event_Record := Event_Record'(Time => Bless_Types.Time'First);
end Event_Record_Store;


subtype Event_Log_Index is Integer range 1 .. Pca_Properties.Event_Log_Size;
type Event_Log is array (Event_Log_Index) of Event_Record;

protected type Event_Log_Store
is
    pragma Priority (10);

    function Get(Ind : in Integer) return Event_Record;
    --# global in Event_Log_Store;

    procedure Put(Ind : in Integer; Val : in Event_Record);
    --# global in out Event_Log_Store;
    --# derives Event_Log_Store from Event_Log_Store, Ind, Val;
private
    TheStoredData : Event_Log := Event_Log'(others => Event_Record'(Time => Bless_Types.Time'First));
end Event_Log_Store;


type Infusion_Type is (Bolus_Infusion, Square_Infusion, Basal_Infusion, KVO_Infusion);

protected type Infusion_Type_Store
is
    pragma Priority (10);

    function Get return Infusion_Type;
    --# global in Infusion_Type_Store;

    procedure Put(X : in Infusion_Type);
    --# global out Infusion_Type_Store;
    --# derives Infusion_Type_Store from X;
private
```

```
        TheStoredData : Infusion_Type := Infusion_Type'First;
    end Infusion_Type_Store;


    type Pump_Fault_Type is (Prime_Failure, Pump_Hot, Bubble, Upstream_Occlusion_Fault,
     Downstream_Occlusion_Fault, Overinfusion, Underinfusion);

    protected type Pump_Fault_Type_Store
    is
        pragma Priority (10);

        function Get return Pump_Fault_Type;
        --# global in Pump_Fault_Type_Store;

        procedure Put(X : in Pump_Fault_Type);
        --# global out Pump_Fault_Type_Store;
        --# derives Pump_Fault_Type_Store from X;
    private
        TheStoredData : Pump_Fault_Type := Pump_Fault_Type'First;
    end Pump_Fault_Type_Store;

end Pca_Types;

package body Pca_Types
is
    protected body Alarm_Type_Store is
        function Get return Alarm_Type
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Alarm_Type)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Alarm_Type_Store;

    protected body Warning_Type_Store is
        function Get return Warning_Type
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Warning_Type)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Warning_Type_Store;

    protected body Status_Type_Store is
        function Get return Status_Type
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
```

```
    end Get;

    procedure Put(X : in Status_Type)
      --# global out TheStoredData;
      --# derives TheStoredData from X;
    is
    begin
        TheStoredData := X;
    end Put;
end Status_Type_Store;

protected body Flow_Rate_Store is
    function Get return Flow_Rate
    --# global in TheStoredData;
    is
    begin
        return TheStoredData;
    end Get;

    procedure Put(X : in Flow_Rate)
      --# global out TheStoredData;
      --# derives TheStoredData from X;
    is
    begin
        TheStoredData := X;
    end Put;
end Flow_Rate_Store;

protected body Drug_Volume_Store is
    function Get return Drug_Volume
    --# global in TheStoredData;
    is
    begin
        return TheStoredData;
    end Get;

    procedure Put(X : in Drug_Volume)
      --# global out TheStoredData;
      --# derives TheStoredData from X;
    is
    begin
        TheStoredData := X;
    end Put;
end Drug_Volume_Store;

protected body Drug_Weight_Store is
    function Get return Drug_Weight
    --# global in TheStoredData;
    is
    begin
        return TheStoredData;
    end Get;

    procedure Put(X : in Drug_Weight)
      --# global out TheStoredData;
      --# derives TheStoredData from X;
    is
    begin
        TheStoredData := X;
    end Put;
end Drug_Weight_Store;

protected body Drug_Concentration_Store is
    function Get return Drug_Concentration
    --# global in TheStoredData;
```

```
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Drug_Concentration)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Drug_Concentration_Store;

    protected body Drug_Record_Store is
        function Get return Drug_Record
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Drug_Record)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Drug_Record_Store;

    protected body Drug_Library_Store is
        function Get(Ind : in Integer) return Drug_Record
        --# global in TheStoredData;
        is
        begin
            return TheStoredData(Ind);
        end Get;

        procedure Put(Ind : in Integer; Val : in Drug_Record)
          --# global in out TheStoredData;
          --# derives TheStoredData from TheStoredData, Ind, Val;
        is
        begin
            TheStoredData(Ind) := Val;
        end Put;
    end Drug_Library_Store;

    protected body Prescription_Store
    is
        function Get return Prescription
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(Prescription_In : in Prescription)
        --# global out TheStoredData;
        --# derives TheStoredData from Prescription_In;
        is
        begin
            TheStoredData := Prescription_In;
        end Put;
    end Prescription_Store;
```

```
protected body Fault_Record_Store is
    function Get return Fault_Record
    --# global in TheStoredData;
    is
    begin
        return TheStoredData;
    end Get;

    procedure Put(X : in Fault_Record)
      --# global out TheStoredData;
      --# derives TheStoredData from X;
    is
    begin
        TheStoredData := X;
    end Put;
end Fault_Record_Store;

protected body Fault_Log_Store is
    function Get(Ind : in Integer) return Fault_Record
    --# global in TheStoredData;
    is
    begin
        return TheStoredData(Ind);
    end Get;

    procedure Put(Ind : in Integer; Val : in Fault_Record)
      --# global in out TheStoredData;
      --# derives TheStoredData from TheStoredData, Ind, Val;
    is
    begin
        TheStoredData(Ind) := Val;
    end Put;
end Fault_Log_Store;

protected body Event_Record_Store is
    function Get return Event_Record
    --# global in TheStoredData;
    is
    begin
        return TheStoredData;
    end Get;

    procedure Put(X : in Event_Record)
      --# global out TheStoredData;
      --# derives TheStoredData from X;
    is
    begin
        TheStoredData := X;
    end Put;
end Event_Record_Store;

protected body Event_Log_Store is
    function Get(Ind : in Integer) return Event_Record
    --# global in TheStoredData;
    is
    begin
        return TheStoredData(Ind);
    end Get;

    procedure Put(Ind : in Integer; Val : in Event_Record)
      --# global in out TheStoredData;
      --# derives TheStoredData from TheStoredData, Ind, Val;
    is
    begin
```

```
            TheStoredData(Ind) := Val;
        end Put;
    end Event_Log_Store;

    protected body Infusion_Type_Store is
        function Get return Infusion_Type
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Infusion_Type)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Infusion_Type_Store;

    protected body Pump_Fault_Type_Store is
        function Get return Pump_Fault_Type
        --# global in TheStoredData;
        is
        begin
            return TheStoredData;
        end Get;

        procedure Put(X : in Pump_Fault_Type)
          --# global out TheStoredData;
          --# derives TheStoredData from X;
        is
        begin
            TheStoredData := X;
        end Put;
    end Pump_Fault_Type_Store;

end Pca_Types;
```

Listing D.4: `Pca_Types` *package*

```
package Pca_Properties
is
    Drug_Library_Size : constant Integer := 500;
    Fault_Log_Size : constant Integer := 150;
    Event_Log_Size : constant Integer := 1500;
    KVO_Rate_Constant : constant Integer := 1;
    KVO_Rate : constant Integer := KVO_Rate_Constant;
    Max_Rate : constant Integer := 10;
end Pca_Properties;
```

Listing D.5: `Pca_Properties` *package*

```
with Pca_Properties,
    Base_Types,
    Bless_Types,
    Ice_Types,
     Pca_Types;
--# inherit Pca_Properties,
--#         Base_Types,
--#         Bless_Types,
--#         Ice_Types,
--#         Pca_Types;
package Pca_Operation
--# own protected Bolus_Duration : Ice_Types.Minute_Store (Priority=>10);
--#     protected Infusion_Flow_Rate : PCA_Types.Flow_Rate_Store (Priority=>10);
--#     protected System_Status : Pca_Types.Status_Type_Store (Priority=>10);
--#     protected Rx : Pca_Types.Prescription_Store (Priority=>10);
--#     task mdphw : Max_Drug_Per_Hour_Watcher;
--#     task rc : Rate_Controller;
--#     task pbc : Patient_Bolus_Checker;
is
    procedure Put_Start_Button_Pressed;

    procedure Put_Stop_Button_Pressed;

    procedure Put_Patient_Request_Bolus;

    procedure Put_Clinician_Request_Bolus;

    procedure Put_Bolus_Duration (Bolus_Duration_In : Ice_Types.Minute);
    --# global out Bolus_Duration;
    --# derives Bolus_Duration from Bolus_Duration_In;

    procedure Get_Infusion_Flow_Rate (Infusion_Flow_Rate_Out : out Pca_Types.Flow_Rate);
    --# global in Infusion_Flow_Rate;
    --# derives Infusion_Flow_Rate_Out from Infusion_Flow_Rate;

    procedure Get_System_Status (System_Status_Out : out Pca_Types.Status_Type);
    --# global in System_Status;
    --# derives System_Status_Out from System_Status;

    procedure Put_Rx (Rx_In : Pca_Types.Prescription);
    --# global out Rx;
    --# derives Rx from Rx_In;


    task type Max_Drug_Per_Hour_Watcher
    --# global in Infusion_Flow_Rate;
    is
        pragma Priority(10);
    end Max_Drug_Per_Hour_Watcher;

    task type Rate_Controller
    --# global in Rx;
    --#        in Bolus_Duration;
    --#        out Infusion_Flow_Rate;
    --#        out System_Status;
    is
        pragma Priority(10);
    end Rate_Controller;

    task type Patient_Bolus_Checker
    is
        pragma Priority(10);
    end Patient_Bolus_Checker;

end Pca_Operation;
```

```
package body Pca_Operation
is
    type la_type is (
                    StopButton,
                    TooMuchJuice,
                    PatientButton,
                    ResumeSquareBolus,
                    ResumeBasal,
                    StartSquareBolus,
                    SquareBolusDone,
                    StartButton);

    Bolus_Duration : Ice_Types.Minute_Store;
    Infusion_Flow_Rate : PCA_Types.Flow_Rate_Store;
    System_Status : Pca_Types.Status_Type_Store;
    Rx : Pca_Types.Prescription_Store;

    mdphw : Max_Drug_Per_Hour_Watcher;
    rc : Rate_Controller;
    pbc : Patient_Bolus_Checker;

    procedure Put_Start_Button_Pressed
    is
    begin
        -- TODO: implement event handler
        null;
    end Put_Start_Button_Pressed;

    procedure Put_Stop_Button_Pressed
    is
    begin
        -- TODO: implement event handler
        null;
    end Put_Stop_Button_Pressed;

    procedure Put_Patient_Request_Bolus
    is
    begin
        -- TODO: implement event handler
        null;
    end Put_Patient_Request_Bolus;

    procedure Put_Clinician_Request_Bolus
    is
    begin
        -- TODO: implement event handler
        null;
    end Put_Clinician_Request_Bolus;

    procedure Put_Bolus_Duration (Bolus_Duration_In : ICE_Types.Minute)
    is
    begin
        Bolus_Duration.Put(Bolus_Duration_In);
    end Put_Bolus_Duration;

    procedure Get_Infusion_Flow_Rate (Infusion_Flow_Rate_Out : out Pca_Types.Flow_Rate)
    is
    begin
        Infusion_Flow_Rate_Out := Infusion_Flow_Rate.Get;
    end Get_Infusion_Flow_Rate;

    procedure Get_System_Status (System_Status_Out : out Pca_Types.Status_Type)
    is
    begin
```

```
        System_Status_Out := System_Status.Get;
    end Get_System_Status;

    procedure Put_Rx (Rx_In : Pca_Types.Prescription)
    is
    begin
        Rx.Put(Rx_In);
    end Put_Rx;


    task body Max_Drug_Per_Hour_Watcher
    is
    begin
        loop
            --# assert PUMP_RATE;
            null;
        end loop;
    end Max_Drug_Per_Hour_Watcher;

    task body Rate_Controller
    is
        la : la_type;
    begin
        loop
            --# assert true;
            --# assert Rx_APPROVED;
            --# assert PUMP_RATE;
            --# assert (la=StopButton) -> HALT;
            --# assert (la=TooMuchJuice) -> KVO_RATE;
            --# assert (la=PatientButton) -> PB_RATE;
            --# assert ((la=StartSquareBolus) or (la=ResumeSquareBolus)) -> CCB_RATE;
            --# assert ((la=StartButton) or (la=ResumeBasal) or (la=SquareBolusDone)) -> BASAL_RATE;
            --# assert (PUMP_RATE = 0) -> HALT;
            --# assert (PUMP_RATE = Pca_Properties.KVO_Rate) -> KVO_RATE;
            --# assert (PUMP_RATE = Patient_Bolus_Rate) -> PB_RATE;
            --# assert (PUMP_RATE = Square_Bolus_Rate) -> CCB_RRATE;
            --# assert (PUMP_RATE = Basal_Rate) -> BASAL_RATE;
            null;
        end loop;
    end Rate_Controller;

    task body Patient_Bolus_Checker
    is
    begin
        loop
            --# assert true;
            null;
        end loop;
    end Patient_Bolus_Checker;

end Pca_Operation;
```

Listing D.6: `Pca_Operation` *package*

Appendix E

PCA Pump - dose monitor module

Final version of `PCA_Verification` and AUnit tests.