Rigorous Methods for Software Engineering (F21RS0):

BSc Coursework 2 (2019-20)

A SPARK 2014 High Integrity Software Development Exercise

Andrew Ireland
Department of Computer Science
School of Mathematical & Computer Sciences
Heriot-Watt University

October 17, 2019

THIS IS AN INDIVIDUAL PROJECT

While discussion with fellow students as to the general nature of this project is acceptable, it is critically important that the solution you adopt as well as the associated code and report are completely your own work. The re-use of other peoples code (other than the code in Appendix A) is not permitted and if identified will be treated as a disciplinary matter. Information on plagiarism can be found via

`https://www.hw.ac.uk/students/studies/examinations/plagiarism.htm`

Contents

1 Introduction

Many engineering facilities make use of water as a coolant. Storage tanks are typically used in order to maintain a sufficient supply of water. Controlling the amount of water within such water tanks can therefore represent a safety critical component of the facility. All such tanks have physical limits which if exceeded will represent a major hazard to life and/or the environment. Automated subsystems which detect such hazards and invoke corrective actions play a critical role in ensuring the overall safety of a system.

The focus of this coursework is the development of software based *Water Tank Protection* (WTP) system. The role of WTP is to ensure that a safe level of water is maintained with the tank. Exceeding the capacity would result in a structural failure in the water tank, which in turn would cause a failure in the water supply. Conversely, if the water level gets too low then the safety of the facilities that it is cooling maybe compromised.

Specifically, your aim is to implement the software component of a simple WTP system using the SPARK 2014 approach to high integrity Ada. You are provided with SPARK package specifications that define the safety-critical boundary of the system as well as a test harness written in Ada. Your task is to develop the system-critical control component as well as the implementation details of the boundary packages. In §2 a system-level description of the WTP system is provided. The software requirements of the exercise are detailed in §3, while the testing requirements are outlined in §4. Finally, in §5 the deliverables that are expected of you are described.

2 System-Level Description

The WTP system comprises of a central controller and 4 boundary subsystems that manage the console, sensors, alarm, and an emergency drainage valve. The sensors measure the level of water within the water tank. In order to reduce the effects of component failure 3 sensors are used. Each sensor generates a value in the integer range 0cm to 2100cm inclusive. The range of valid water level values is divided into 3 categories:

LOW : value between 0cm and 99cm.

NORMAL : value between 100cm and 1999cm.

HIGH : value between 2000cm and 2099cm.

If a sensor reading exceeds 2099cm, i.e. a value of 2100cm, then it is deemed to be undefined (**UNDEF**). The sensor value returned to the WTP controller is calculated as the majority of the 3 sensor readings. For example, if two sensors give a reading of 99cm while the third gives a reading of 100cm, then the majority value is 99cm. If there does not exist a majority then an undefined value is returned to the WTP controller. If a **NORMAL** sensor value is returned to the WTP controller, and the alarm is not enabled, then no action is required. If a **HIGH** sensor value is returned to the WTP controller then the alarm is enabled. Once the alarm is enabled, if subsequent sensor readings do not show a decrease in the water-level then the WTP controller should open the emergency drainage valve. However, if subsequent sensor readings show a decrease in the water-level then the alarm should be disabled when the water-level returns to **NORMAL**. If a **LOW** sensor value is returned to the WTP controller then the alarm is enabled. Once the alarm is enabled, if the water-level eventually returns to **NORMAL** then the alarm should be disabled. If a reset request is received by the WTP controller, and the water-level is below the **HIGH** threshold,

then the WTP controller should ensure that the emergency drainage valve is closed and that the alarm is disabled.

3 Software Requirements

The safety-critical software for the WTP system is to be written in SPARK. The safety-critical functionality is to be distributed across 5 packages (subsystems) as described below:

`Console`**:** provides an interface to the WTP controller. Specifically it supports a reset mechanism and is responsible for maintaining the state information on the reset subsystem.

`Sensors`**:** responsible for maintaining and providing access to the current values of the sensors.

`Alarm`**:** provides control of the WTP alarm and is responsible for maintaining the state information on the alarm subsystem. In addition it also counts the number of alarm events.

`Drain`**:** provides control of the water tank's emergency drainage valve and is responsible for maintaining the state information on the Drain subsystem.

`WTP`**:** responsible for the overall control provided by the WTP system.

SPARK package specifications are given in Appendix A for `Console`, `Sensors`, `Alarm`, `Drain` Note that the actual code for these package specifications can be obtained from:

```
http://www.macs.hw.ac.uk/~air/rmse/code/WTP/
```

You are required to:

R1: Develop package bodies consistent with the specifications given in Appendix A. You should use state abstraction and refinement (*i.e.* `Abstract_State` and `Refined_State`) in order to **promote information hiding**. This will involve the use of the `Refined_Global` and `Refined_Depends` aspects. In addition, you should use comments within your code to explain the role of variables and key decision points, etc.

R2: Develop a `WTP` package (`.ads` and `.adb`) that implements the intended behaviour of the `WTP` controller procedure, i.e. `WTP.control`, as described in §2. When writing the contracts you should make use of state abstraction and refinement (*i.e.* `Abstract_State` and `Refined_State`) in order to **promote information hiding**. Again, you should use comments within your code to explain the role of variables and key decision points, etc.

R3: The `WTP` package must be consistent with the 4 given package specifications as well as the test harness described below in §4.

R4: Using the SPARK verification tool (i.e. `GNATprove` flow mode), your SPARK code should be shown to be free of data and information flow analysis errors.

R5: Using the SPARK verification tool (i.e. `GNATprove` prove mode), your SPARK code should be shown to be free from run-time exceptions. Hint: this will typically involve using preconditions and defensive programming.

4 Software Testing Requirements

You are given a test harness for the purposes of testing your WTP system software. The test harness is written in Ada (i.e. not SPARK compliant) and allows the simulation of the environment within which the WTP system is intended to operate. The test harness involves 2 packages that are described as follows:

`Handler`**:** responsible for:

1. maintaining the state information associated with the `Sensors` and `Console` packages. The state information is read from a file called `env.dat` (see Appendix B.1).
2. logging the state information associated with `Sensors`, `Alarm`, `Drain`, `Console` as well as the majority sensor reading (category). The logger writes to a file called `log.dat` (see Appendix B.2).

`Test_WTP`**:** responsible for running *your* `WTP` control procedure against test data (i.e. `env.dat`). The actual reading of the test data and the recording of the results (i.e. `log.dat`) is performed by subprograms defined within the `Handler`.

Note that the code for these 2 packages is provided in:

`http://www.macs.hw.ac.uk/~air/rmse/code/WTP/`

Example `env.dat` and `log.dat` files are also given in this directory.

5 Deliverables

Your submission via VISION **should** take the form of i) a report and, ii) a zip file containing all SPARK code related to your solution. Your report **must** include the following:

D1: A statement of any assumptions you have made about the requirements, and how they have impacted on your implementation decisions. *[3-marks]*

D2: A diagrammatic representation of the software architecture of your WTP system software. Your diagram should communicate the subsystems and their local state, as well as their connectivity, *i.e.* imports and exports. *[4-marks]*

D3: Listings of each source file, i.e. **ALL 5 packages** (specifications & bodies) that define the WTP system software (see requirements). Ensure that your source files are well formatted and are readable. *[18-marks]*

D4: The summary file generated by the SPARK analysis of your code, i.e. the use of `gnatprove.out` in performing flow analysis and in proving exception freedom. *[1-mark]*

D5: The `log.dat` file generated by your WTP system software for the `env.dat` file given in Appendix B.1. *[1-mark]*

D6: The `log.dat` file generated by your WTP system software for the `env.dat` file given in: `http://www.macs.hw.ac.uk/~air/rmse/assignment/env.dat` **Note the above test data (D6) will be made available at the start of week 10.** *[1-mark]*

D7: With explicit reference to your code, describe how you protected against run-time exceptions, i.e. what you protected against and how you achieved the protection. *[4-marks]*

There will be 4-marks allocated according to the quality of your report. *[36 marks in total]*

6 Submission Deadline

Your submission (report and code) should be uploaded to VISION by 3.30pm (local time) on December 6, 2018 (end of week 12).

Note that this is an individual project which means that your submission MUST be your own work. This assignment counts for 50% of the coursework marks that are available for this course.

The University Policy on the Submission of Coursework can be found via the following link:

`https://www.hw.ac.uk/services/docs/learning-teaching/policies/submissionofcoursework-policy.pdf`

7 Feedback

You will receive feedback on your submission within 15 working days.

A SPARK Package Specifications (Appendix)

Presented below are the SPARK package specifications for `Console`, `Sensors`, `Alarm`, `Drain`.

A.1: `console.ads`

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       13.9.2019
--
-- Filename:            console.ads
--
-- Description:         Models the console associated with the WTP system, i.e.
--                      the reset mechanism that is required to disable the
--                      escape valve subsystem.

pragma SPARK_Mode (On);
package  Console
with
Abstract_State => State
is
   procedure Enable_Reset
   with
     Global => (Output => State),
     Depends => (State => null);

   procedure Disable_Reset
   with
     Global => (Output => State),
     Depends => (State => null);

   function Reset_Enabled return Boolean
   with
     Global => (Input => State),
     Depends => (Reset_Enabled'Result => State);
end Console;
```

A.2: sensors.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       13.9.2019
--
-- Filename:            sensors.ads
--
-- Description:         Models the 3 pressure sensors associated with the WTP system.
--                      Note that a single sensor reading is calculated using a majority
--                      vote algorithm.

pragma SPARK_Mode (On);
package Sensors
with
Abstract_State => State

is
   subtype Sensor_Type is Integer range 0..2100;
   subtype Sensor_Index_Type is Integer range 1..3;

   procedure Write_Sensors(Value_1, Value_2, Value_3: in Sensor_Type)
   with
     Global => (Output => State),
     Depends => (State => (Value_1, Value_2, Value_3));

   function Read_Sensor(Sensor_Index: in Sensor_Index_Type) return Sensor_Type
   with
     Global  => (Input => State),
     Depends => (Read_Sensor'Result => (State, Sensor_Index));

   function Read_Sensor_Majority return Sensor_Type
   with
     Global => (Input => State),
     Depends => (Read_Sensor_Majority'Result => State);
end Sensors;
```

A.3: alarm.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       13.9.2019
--
-- Filename:            alarm.ads
--
-- Description:         Models the alarm device associated with
--                      the WTP controller and the alarm count.


pragma SPARK_Mode (On);
```

```
package Alarm
with
Abstract_State => State
is
   procedure Enable
   with
     Global  => (In_Out => State),
     Depends => (State  => State);

   procedure Disable
   with
     Global  => (In_Out => State),
     Depends => (State  => State);

   function Enabled return Boolean
   with
     Global => (Input => State),
     Depends => (Enabled'Result => State);

   function Alarm_Cnt_Value return Integer
   with
     Global => (Input => State),
     Depends => (Alarm_Cnt_Value'Result => State);
end Alarm;
```

A.4: Drain.ads

```
-- Author:              A. Ireland
--
-- Address:             School Mathematical & Computer Sciences
--                      Heriot-Watt University
--                      Edinburgh, EH14 4AS
--
-- E-mail:              a.ireland@hw.ac.uk
--
-- Last modified:       13.9.2019
--
-- Filename:            drain.ads
--
-- Description:         Models the emergency drainage valve associated with the
--                      water tank.

pragma SPARK_Mode (On);
package Drain
with
Abstract_State => State
is
   procedure Open
   with
     Global => (Output => State),
     Depends => (State => null);

   procedure Close
   with
     Global => (Output => State),
     Depends => (State => null);

   function Openned return Boolean
   with
     Global => (Input => State),
     Depends => (Openned'Result => State);
end Drain;
```

B Example Test Data & Results (Appendix)

B.1: An example `env.dat` file

The data within `env.dat` is used to simulate the values of the state variables associated with the boundary packages of the WTP system software. Each row represents the set of values at a particular point in time. The first row denotes the initial set of values in the test run while the last row denotes the final set of values. Columns 1 to 3 denote the values associated with the 3 sensors as described in §2. Column 4 provides the state of the reset mechanism, *i.e.* where 0 denotes reset disabled and 1 denotes reset enabled.

```
 100  100  100 0
1999 1999 1999 0
2001 2001 2001 0
2001 2001 2001 0
2010 2010 2010 0
2010 2010 2010 0
2010 2010 2010 1
2010 2010 2010 1
1900 1900 1900 1
1900 1900 1900 0
1900 1900 2000 0
1900 1900 2000 0
1900 1900 1900 0
```

B.2: An example `log.dat` file

The data within `log.dat` is generated by the logger (see `Handler`). It records state and related information across a simulated test run of the WTP system software. The particular file shown below was generated using the `env.dat` file given in B.1. Note that there are double the number of entries in `log.dat` as there are in `env.dat`. This is because the logger is invoked twice within `Test_WTP`, *i.e.* before and after each invocation of `WTP.Control`.

SENSOR-1	SENSOR-2	SENSOR-3	MAJORITY	ALARM	DRAIN	RESET	ALARM_CNT
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	0
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	0
HIGH	HIGH	HIGH	HIGH	--	CLOSED	--	0
HIGH	HIGH	HIGH	HIGH	ON	CLOSED	--	1
HIGH	HIGH	HIGH	HIGH	ON	CLOSED	--	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	--	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	--	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	--	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	--	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	--	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	ON	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	ON	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	ON	1
HIGH	HIGH	HIGH	HIGH	ON	OPEN	ON	1
NORMAL	NORMAL	NORMAL	NORMAL	ON	OPEN	ON	1
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	ON	1
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	HIGH	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	HIGH	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	HIGH	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	HIGH	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	1
NORMAL	NORMAL	NORMAL	NORMAL	--	CLOSED	--	1