Development & Formal Proof of a Submarine Controller System

Svetlozar Georgiev

SET10112 Formal Approaches to Software Engineering

Abstract. This report describes the implementation of a submarine controller system in the Ada programming language and its formal verification with the utilities of SPARK. The system contains the following subsystems: oxygen control, airlock control, nuclear reactor control, and weapon control.
The finished system implements all features described in the specification and it passes SPARK's Gold level. The design principle used is Object Oriented programming. An extra feature added is a command line interface which displays a simulation of the operation of the submarine.

1 Introduction

The aim of this report is to describe the implementation and formal proof of a nuclear submarine control system. It controls the following subsystems:

- The airlocks.
- The oxygen subsystem.
- The nuclear reactor.
- The weapon subsystem.

The programming language Ada was used for the development of the system, while SPARK was used for the formal proof of the functionality. Extensions were added to the initial specification: Object Oriented Programming was used for the development and a simulation of the functionality of the submarine was added in the form of a command line interface.

2 Controller Structure

The system was divided into packages based on each subsystem. This allowed the separation of code based on its purpose. The following packages were implemented:

- Airlocks - defines an airlock of a submarine which consists of an array of `Door` records. They contain a status (indicating whether the door is *Open* or *Closed*) and a `lock` (signifying whether the door is *Locked* or *Unlocked*). The implemented procedures allow the opening and closing of doors as well as locking and unlocking. Additionally, the status of individual doors can be retrieved with the functions added.

- Oxygen Controller Subsystem - implements the oxygen control system which tracks the levels of oxygen available at all times. It was implemented as a `record` with properties for the oxygen level of the submarine and the status of the system. A procedure was added to set the status of the system to *Critical* when the oxygen levels are too low (so that other subsystems can be notified).
- Reactor Controller Subsystem - the nuclear reactor of the submarine. The system tracks its temperature and status at all times. It is defined as a `record` with properties for the current temperature of the reactor and its status. The status is updated based on the temperature and it is set to *Overheated* when the temperature is too high. This is neeeded so that the submarine can resurface in order to continue to operate safely.
- Weapon Subsystem - the weapon system of the submarine which tracks the number of torpedoes available. The procedures implmented in it allow the storage and firing of torpedoes.
- Submarine - defines a submarine object which incorporates all the subsystems mentioned above.

Separating the code into suitable packages allowed for faster development and better maintainability.

Global variables were not used, instead `record` properties were implemented and necessary values were passed to functions as parameters. The only exception is in the `main` package where an instance of the `Submarine` record is required to carry out the simulation.

3 Descriptions of procedures and functions

This section will describe each subsystem in more detail.

3.1 Airlocks

The airlock system is crucial to the safety of the submarine. It is implemented as a separate package.

An airlock is defined as an array of `Door` records. A door has 2 attributes: `status` and `lock`. The former refers to whether the door is *Open* or *Closed*. The latter indicates whether it is *Locked* or *Unlocked*. Both attributes are implemented as a `type` with 2 possible values. Since the requirement stated that a number of doors have to be closed at all times, procedures to close and open the doors were added. The preconditions for the procedure to close the doors checks whether there exists at least one door that is *Open*, while the postcondition ensures that all doors are *Closed*.

The `LockDoors` procedure has similar pre- and postconditions, however they check the locks of the doors instead of the status.

Lastly, invariant functions which return the status of the doors were implemented so that they can be used in the preconditions in other subsystems since the submarine cannot operate unless the airlock doors are closed and locked.

A function which creates, initialises and returns an airlock record was also added to act as a constructor for the package.

3.2 Oxygen Control

The oxygen control system (`OxygenTank`) is also implemented as a record. It has 2 attributes which indicate the current amount of available oxygen and the status of the system.

The amount of oxygen is stored in a variable of type `O2Level` which ensures its value is between 0 and 100 (indicating percentage).

The status is set based on the percentage of available oxygen and its possible values are *Sufficient* and *Critical*. Therefore, other systems can query the oxygen system and act accordingly based on its status.

A procedure (`UpdateO2Level`) to update the current level of oxygen which takes a `OxygenTank` record as an `in out` parameter and a value for the new level as an `in` parameter was implemented. Its precondition ensures that the new value for the amount of oxygen passed is within the valid range. Its postcondition is implemented as a Contract Case and it checks whether the status of the system is set to the correct value based on the amount of available oxygen. That is, when the oxygen level is higher than a threshold (stored in a constant variable), the status should be *Sufficient*. On the other hand, when its lower, it should be set to Critical.

Another procedure (`UpdateO2Status`) was implemented to Update the status of the system. It is used within `UpdateO2Level`. This procedure was made `private` since it is only used within the package and it does not need to be accessed outside of it. Its precondition checks whether the value for oxygen is within the allowed range. In addition, the postcondition makes sure that the status is set correctly using a similar Contract Case.

Lastly, a function to generate an instance of an oxygen control system was created to act as a constructor for the class.

3.3 Reactor

This subsystem controls the nuclear reactor of the submarine. The reactor is represented as a `record` with 2 attributes: `temperature` and `status`. The former contains the value of the current temperature of the reactor, which increases as the submarine's depth increases. The latter is the status of the reactor, i.e. *Running* when the temperature is below a threshold and *Overheated* when it is above the threshold. The status is used to notify other systems in the event that the temperature is too high.

The temperature is a variable of a custom type which specifies a range for valid values (between 0 and 100). Furthermore, the status is also a custom type. The values it can accept are either *Running* or *Overheated*. In addition, The threshold, which defines when the reactor has overheated, is specified as a `constant` variable of the same type as the temperature variable.

This package also includes several procedures. Firstly, a `private` procedure to update the status of the reactor was added. It takes a reactor record as an `in out` parameter. It then checks its current temperature and updates the status accordingly. The precondition for this function checks whether the reactor's temperature is within the valid range. Its postcondition is a Contract Case which checks that for values below and above the threshold the status is set to Running and Overheated respectively.

Another procedure added was `UpdateTemperature`. It takes a Reactor record an an `in out` parameter and a temperature value as an `in` parameter and changes the current temperature of the reactor to the passed value. Lastly, it calls the `UpdateStatus` functions so that the status reflects the change in temperature. The precondition for this operation is that the passed value is within the valid range for temperatures. The postcondition, which is a Contract Case, ensures that the status is set correctly when the value is below and above the temperature threshold.

The last function in this package constructs a reactor record and returns it similarly to a constructor in other programming languages.

3.4 Weapon System

The weapon system package provides a `Weapon` record. It contains a `torpedoes` attribute. It is a variable of a custom type, `TorpedoRange`, which specifies its valid range, i.e. the number of torpedoes that can exist on the submarine. Additionally, the weapon record contains a `loaded` boolean which indicates whether the weapon is ready to fire.

Three procedures were implemented in this package. Firstly, `Store` takes a `Weapon` record as an `in out` parameter. It then checks whether the current number of torpedoes is below the maximum allowed value. If this is true, the number of torpedoes is incremented. The precondition for this operation is that the number of torpedoes is less than the maximum allowed number. The postcondition checks whether the number has been incremented by exactly 1.

Another procedure, `Load`, also takes a Weapon record as an `in out` parameter. It then sets its `loaded` variable to True if it is not loaded already. The precondition for this function ensures that the weapon is not already loaded and that its torpedo count is greater than 0. Its postcondition checks whether `loaded` has been successfully set to `True` and that the number of torpedoes is the same as before it was loaded.

The last procedure in this package is `Fire`. It takes a `Weapon` as an `in out` parameter. If the weapon is loaded and it has more than 0 torpedoes available, it decreases its torpedo count by one and sets it's loaded variable to `False`. The precondition for this procedure checks whether the weapon is loaded and it has available torpedoes. The postcondition checks if the loaded variable has successfully been set to `False` and that the number of torpedoes has decreased by exactly 1.

A function to create, initialise and return a `Weapon` record was also added to be used as a constructor.

3.5 Submarine

The submarine combines all implemented packages. It implements a `Submarine` record which has the following attributes:

- `position` - the x and z coordinates of the submarine which indicate its current position along the respective axes. It is a variable of a custom type which is an array of 2 Integers.
- `current_depth` - the current depth position of the submarine. It can be considered its y coordinate. Its type is a range between 0 and 500.
- `airlock` - a variable of type `Airlock`.
- `oxygen_tank` - a variable of type `OxygenTank`.
- `reactor` - a variable of type `Reactor`.
- `weapon` - a variable of type `Weapon`.

Invariants were added to this package to simplify the usage of often checked conditions. Firstly, a function which returns a boolean based on whether both airlock doors are *Closed* and *Locked* was created. The second invariant ensures the current depth of the submarine is within the valid range.

The procedures of this package are `Move` and `EmergencySurface`. `Move` takes a Submarine record as an `in out` parameter and a new depth value as an `in` parameter. After validating the new depth is within the allowed range, it updates the current position of the submarine. The precondition for this procedure checks whether the doors of the airlock are closed and locked using the invariant described above. It also checks whether the new depth is within the valid range and ensures there are sufficient levels of oxygen and that the reactor has not overheated. The postcondition ensures that the new position is within the valid range.

`EmergencySurface` takes a `Submarine` record as an `in out` parameter. It then changes its depth to 0 so that the submarine resurfaces. The precondition for this procedure checks whether the status of the oxygen system is *Critical* or the status of the reactor is *Overheated*. Either one of these conditions must be true as well as the invariant which checks whether the airlock doors are closed and locked. The postcondition ensures that the new depth is exactly 0.

3.6 Simulation

As an extra feature, a command line interface was added to show the current status of the submarine. This was achieved by printing the status of each subsystem. Additionally, `Update` procedures were added to the Submarine, SubmarineOxygenTank and SubmarineReactor records.

The Update methods of the reactor and the oxygen system simply modify the values of the systems by a fixed amount if the submarine is not at surface level. For the oxygen system, the level of oxygen is reduced while for the reactor the temperature is increased. These methods are called in the submarine's Update method. The status of both systems is then checked. If either one is critical, the

EmergencySurface method is called so that the submarine can regain oxygen or decrease the temperature of the reactor.

Lastly, a main package was added which creates a global submarine variable. An infinite loop is then started, which updates the submarine and prints its status at each iteration.

4 Proof of Consistency

An important operation of the submarine is the movement procedure. It has to check whether the submarine is not diving below a certain depth. To do that, it has to ensure that the new depth position is correct. This is achieved by proving that the difference between the new position and the current position is exactly the desired amount to move. If *current* is the current position of the submarine, and $current + desired$ is the new position, then:

$$new - current = (current + desired) - current = desired$$

Therefore, the amount of units the position was changed by is *desired* and the movement was correct.

Another important part of the movement of the submarine is the EmergencySurface procedure. If the doors of the submarine are closed and locked and the oxygen system status is critical or the reactor system's status is overheated, then the position should be 0 and the statuses should change to sufficient and running respectively. A proof is provided below. P_1 is the status of the oxygen system, Q_1 is the status of the reactor, pos_1 is the current position, P_2 is the status of the oxygen system after resurfacing, Q_2 is the status of the reactor after resurfacing, pos_2 is the new position.

$$\begin{array}{c}
Ax \qquad\qquad\qquad\qquad Ax \\ \hline
P_1, pos_1 \implies P_2, Q_2, pos_2 \qquad Q_1, pos_1 \implies P_2, Q_2, pos_2 \\ \hline
P_1, pos_1 \implies P_2, Q_2, pos_2 \qquad Q_1, pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \\ \hline
P_1, pos_1 \implies P_2, Q_2, pos_2 \qquad Q_1 \wedge pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \\ \hline
P_1, pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \qquad Q_1 \wedge pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \\ \hline
P_1 \wedge pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \qquad Q_1 \wedge pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \\ \hline
(P_1 \vee Q_1) \wedge pos_1 \implies P_2 \wedge Q_2 \wedge pos_2 \\ \hline
\implies (P_1 \vee Q_1) \wedge pos_1 \supset P_2 \wedge Q_2 \wedge pos_2
\end{array}$$

Another verified part of the program is the Weapon system. It's verification process ensures that if a torpedo is stored, the number of torpedos has increased by 1.

Let t be the current number of torpedoes and $t + 1$ be the number after adding 1 torpedo. It can then be calculated that the difference between the new number of torpedos and their number before adding a new one is:

$$t_{new} - t_{old} = (t + 1) - t = 1$$

Therefore, the number of torpedoes has increased by exactly 1.

5 Conclusion

To conclude, the specified requirements were implemented. Additionally, extensions were added in the form of a simulation of the operation of the submarine. Furthemore, the software was designed with Object-Oriented principles. The functionality of the resulting system was proved with SPARK and it achieved Gold level with some exceptions. Firstly, warning are shown due to the use of Ada's text printing functions. Secondly, two procedures' preconditions are considered weak (the update procedures of the oxygen and reactor subsystems) due to their usage in the submarine's update procedure. Overall, it can be said that the requirements were met and a satisfactory level of proof was achieved.

Future work could include user input to control the submarine in the simulation or a graphical user interface could be added. In addition, the torpedoes can be implemented as a record with suitable attributes. Lastly, the submarine airlocks can be automatically unlocked when at surface level and locked when diving.