Implementation of submarine control system with safety critical procedure

Alessio Williams Gava - 40212064

Napier University, Formal Approaches to Software Engineering - SET10112 2018-2019

1 Introduction

This report is aimed to the describe the SPARK implementation of a submarine control system. The application is focused on the moving and the constraints aspects of the submarine utilize. More specifically, it deals with the submersion, horizontal movement, resurface and constraints related to the reactor temperature and oxygen level. It is a simplistic high abstraction of the submarine functionalities, which is interested in the main activities, taking for granted many aspects on which the main functions are built upon.
It has been used the SPARK language to ensure the correct functionality of the implemented submarine's processes, conforming to the contracts set for each method created (preconditions-postconditions, predicates, etc).

2 Controller Structure

The submarine controller has been split in different components which are in charge of its various aspects: oxygen, reactor, airlock doors and the main controller, submarine, which controls the movement functionalities and the structure of the submarine record.

Oxygen controller: Used to control the oxygen levels in the submarine and keep track of its status issuing a warning in case it gets too low.
A main type is created to describe the oxygen, which is the *OxygenLv* type 0..100. This is the percentage of available oxygen in the tanks and it can be used for any type of submarine. Two global variables are utilized to describe the percentage of Low Oxygen (*OxygenLow*) and the constant use of oxygen per distance moved, which could be different from submarine type (depending on their tank size).

Reactor controller: Used to supervise the reactor temperature while the submarine is on movement. The main type created is the *ReactorTemp* which is used to describe the range of temperature the reactor could reach. Again, two global variables are available, the first to describe the point in which the reactor is considered overheat (*ReactorOverheat)* and the second to determine the increase of temperature per distance moved. Compared to the oxygen levels, the reactor is not very suitable in utiliz-

ing a percentage type because there is not a specific maximum and minimum temperature.

Air door controller: Used to describe and control the air doors on the submarine.
A record *Door* is utilized to bundle the possible states in which a door can be, position and lock. Both of them have their own types, *DoorLock(Locked, Unlocked)* and *DoorPosition(Open, Closed)*, restraining to the only values that the door can be set.
Finally, an array is declared which stores a number of Doors, describing the ones present in the submarine. An helper function is present to return an array of specified number of doors, which can then be set into the submarine record.

Submarine controller: Main controller that comprises information from the other sub-controllers.
The record collects all the necessary members to describe the submarine at a minimum level to run the chosen functionalities. It is presented as follow:

```
type Submarine is
      record
         status : SubStatus := Submerged;
         doors_status : Doors := (0=> (Open, Unlocked),
1=> (Open, Unlocked));
         oxygen : OxygenLv := OxygenLv'Last;
         reactor : ReactorTemp := ReactorTemp'First;
         actual_depth : Depth := Depth’First;
         actual_horizontal_position : Integer := 0;

      end record;
```

As it can be seen, the record is a new type Submarine which is composed by other units, provided by the correct packages that are focused on dealing with that topic. This is to mimic OOP techniques, enforcing encapsulation and lowering decoupling.
Most of the items are of a non-primitive type, except for the horizontal distance, Integer. Each type has been created to respect the constraint given by the simulation environment, so when a variable of that type is initialized or modified, it can’t be outside the range specified. In fact, *Depth* is utilized to describe the maximum Depth reachable from the submarine. For the horizontal position is has been used an Integer because there is no constraint on the distance where it can be, and it is possible to use negative distances to mimic a reverse movement. Another type created in this controller is the *SubStatus* (Submerged, InWater) which specify if submarine is in water or not.
The record members are all default initialized, providing with the correct properties set for a submarine submerged with two air doors (it can be modified using helper function in the Air Door package). Additionally, they can be initialized with different values to describe another type of submarine.

3 Description of procedures and functions

In this section is further analyzed each controller and presented their functions, procedures and subprogram contracts. Types used have been already introduced in previous section.

Oxygen Controller: Only needed procedures to achieve the submarine functionalities have been implemented.
In this package are set the constant variables *OxygenLow* to 20, indicating that at 20% the submarine is at critical level of oxygen. The *oxygen_usage_per_distance* is simply set to 1, meaning that for each submarine step moved, it utilize a 1% of oxygen.

- OxygenInvariant: function invariant which returns a Boolean used to check if the oxygen level in the submarine is in the safe range (> *OxygenLv'First* and < *OxygenLv'Last*). It is used from other submarine functions to decide for a possible emergency emersion. Additionally, being an invariant is used on selected moving functionalities procedures condition.
- FullChargeOxygen(oxygen : in out OxygenLv): sets the parameter oxygen to full level. The only condition is the Post where the oxygen level replenished has to be equal to the maximum value of the oxygen type.
- ReduceOxygen(oxygen : in out OxygenLv): procedure which reduces the passed *oxygen* parameter utilizing the constant variable *oxygen_usage_per_distance*. The only Post conditions are that the modified oxygen is lower than the original and that is not lower than the minimum. It has been used < rather than =, to prevent the increase of oxygen which would be gained in case the *oxygen_usage_per_distance* is set as a negative value. This is done in case a future implementation will allow negative value, but at the moment is not.
- ShowWarning(oxygen : in OxygenLv): simple procedure that issues a warning of low oxygen, printing also the oxygen level. No Post conditions are needed because there is no modification of the data inputted (note the use of *in* operator), but it can be useful to set a Pre-condition that checks the oxygen level is lower than the OxygenLow threshold.

Reactor controller:
- ReactorInvariant: function invariant which returns a Boolean, checking that the temperature passed as argument is not higher than the ReactorOverheat temperature. It is used from other submarine functions to decide for a possible emergency emersion. Additionally, being an invariant is used on selected moving functionalities procedures conditions.
- FullDecreaseTemperature(temperature : in out ReactorTemp): decrease to the minimum temperature the parameter passed. The Post condition is that the temperature has to be equal to the ReactorTemp'First range.

- IncreaseTemperature(temperature : in out ReactorTemp): increases the *temperature* passed as argument of an amount which is set from the constant *increase_temp_per_distance*.
  The Post conditions are that the resulting temperature must not be higher or equal than the minimum set from the *ReactorTemp* range and strictly lower than the old temperature. This is to ensure that a decrease has indeed been applied.

Air Door Controller: the main type used is the *Door* which sets the two properties position and lock. An array *Doors*, is created and as mentioned can be set to any positive integer range index, returning an array of wanted number of *Door*.

- OneDoorClosedInvariant(doors : in out Doors): function returning Boolean that checks a door is always closed in the array of *Door*. This is needed from submarine specification to be able to operate. Having dynamic array size, is needed to be checked all the doors for at least one closed.
- DoorsClosedLockedInvariant(doors : in out Doors): second invariant needed to satisfy the condition that a submarine cannot undertake any movements without having all the doors closed. It returns a Boolean, True in case all the doors position = Closed and lock = Locked. This is a stronger invariant compared with the previous one and does not need to be satisfied when submarine is submerged, only when performing actions underwater.

The following procedures only implemented certain combination of setters for the doors in a way that is not possible to set doors with erroneous combination. For example, it will not be possible to set a door as *Open* and *Locked*.

- CloseLockDoors(doors : in out Doors): procedure to first close and then lock all doors. The Pre condition is that at least one door is closed and the Post is that all doors are locked and closed (satisfy second invariant).
- UnlockDoors(doors : in out Doors): procedure that unlocks all the door. This is used when the submarine has submerged and is preparing to open the doors (except one), it will need first to unlock all of them. Since the doors are being unlocked, the submarine will not be underwater, so the Pre condition is *OneDoorClosedInvariant*. The Post is that all the doors *lock = Unlocked*.
- OpenSpecificDoor(doors :in out Doors, index : in Natural): procedure that opens only a required door at a time. Using it, can be decided which doors to let open or closed. It will first unlock and then open, so it does not matter if the door was already unlocked or not. This procedure can be applied only when out of water. No Pre condition is needed since as Post is set the *OneDoorClosedInvariant* and if it is holding after opening one door it has to hold already before opening it. The second Pre condition is that the door at the index provided is *lock = Unlocked* and *position = Open*.

Submarine Controller: all the moving operations are implemented in this package. In this simulation the submarine can only move in a 2-dimensional space because a

third dimension is not set in the submarine record and a Rotate procedure is not provided.

The moving operations are divided in vertical and horizontal. The vertical is additionally subdivided in ascending and descending because there are differences in performed inner actions. When moving vertically towards the top, the oxygen and reactor levels are not modified. Instead, when it is towards the bottom, the oxygen is reduced. When moving horizontally, only the reactor temperature is increased.

Each operation has to guarantee that the *DoorsClosedLockedInvariant* is respected before being able to apply it. Other than utilize Pre condition, it has been set also in the code that the method can only be executed in case of adherence of the invariant (using an initial if statement for each method).

- Resurface(submarine : in out Submarine): utilized to get the submarine at lowest depth (*Depth'First*) in case of emergency ascension. Once it reaches the top, doors will be unlocked, opened and oxygen will be replenished. Additionally, the temperature of the reactor is full decreased.

  Pre condition is the *DoorsClosedLockedInvariant* or it will not be able to perform the actions. The oxygen and reactor invariants cannot be satisfied because they are the reason that this procedure is executed. They are instead, present in the Post condition, because their levels are stabilized when *Submerged*. Additionally, the *actual_depth* of the submarine has to be the lowest depth (*Depth'First*).
- Dive(submarine : in out Submarine; to_depth : in Depth): procedure utilized to move the submarine in a vertical fashion, towards the bottom. It can be used either from a status of *Submerged* or when the submarine is in the Water (*InWater* status). The *to_depth* argument is used to move the submarine to that specified *Depth*. Each step moved will trigger a reduction of oxygen that is calculated by the oxygen package and returned. In case the *to_depth* is higher than the current depth, the submarine will not perform the action. Additionally, the maximum depth cannot be surpassed, so the submarine will go no further. A check is performed after each step moved, using the *OxygenInvariant*, controlling that the oxygen is not finished. In case it is, an emergency emersion is performed with the utilize of a different procedure: *Resurface*.

  This procedure has multiple possible starting statuses: *InWater* or *Submerged*. Nonetheless, the Pre condition has to satisfy the *DoorsClosedLockedInvariant* as already stated. The *OneDoorClosedInvariant* is satisfied because it is a subset of the previous invariant. Additionally, *OxygenInvariant* and *ReactorInvariant* are set as Pre or it will not be able to start diving; it does not mean that they need to be full, only not over the critical threshold.

  The Post condition still have to guarantee the oxygen and reactor invariants and that it did not descended lower than the maximum depth (*Depth'Last*), but also take into account that the submarine could have performed the emergency emersion. This is dealt with a case statement: in case the status = *Submerged*, it means that the *Resurface* has been applied and its same Post conditions are applied: *actual_depth* is at the lowest (*Depth'First*) and *OneDoorClosedInvariant* because the doors have been opened to load the oxygen. In case the status = *Submerged*, it

means that submarine is still in water so the *DoorsClosedLockedInvariant* needs to be satisfied.

- Ascend(submarine : in out Submarine; to_depth : in Depth): used to move the submarine vertically toward the top of the water, and if it reaches it, status *Submerged* is set.
  Pre condition are *DoorsClosedLockedInvariant* to be able to operate, and that the *actual_depth* is higher than *to_depth* (moving up), oxygen and reactor invariants are needed as well.
  Post conditions are created as a case statement because in case that the status is *Submerged*, only the *OneDoorClosedInvariant* is needed. In case the submarine is in water, *DoorsClosedLockedInvariant* and the oxygen and reactor invariants are to be respected.
- MoveHorizontal(submarine : in out Submarine; distance : Integer): procedure used to move the submarine of a desired distance (can be positive or negative). For each step moved, the temperature of the reactor is increased. In case it is more than the *ReactorOverheat*, a resurface is operated by the procedure *Resurface*.
  Pre conditions are the usual *DoorsClosedLockedInvariant* and the oxygen and reactor invariants.
  Post conditions utilize the same use case explained in the *Ascend* and additionally is needed that the new position is the sum of the old position with the distance inputted.

4 Proof of consistency

Proof of consistency has been achieved with the SPARK proving mechanism which utilizes the Pre, Post conditions along with the variable range types to check that the conditions are respected for any procedures and functions.

5 Conclusion

In conclusion, for this simplistic submarine control system, most of the functionalities have been implemented at exception of the torpedoes loading/storing/firing. SPARK proving at Gold level; in a few cases cannot prove preconditions, mostly because of the utilize of procedures inside other procedures, not matching their preconditions.
As future works it would be needed to modify the increasing of oxygen and temperature per distance moved. At the moment they are linearly increased for each step moved. It would be better to follow a function that is based on different variables as depth, current temperature, etc.. The variables utilized are using only integer numbers, it would be better to utilize floating point values to have an increased precision (Depth, OxygenLv). Some of the checks and Post conditions will have to be modified to reflect the increased the conditions.
The properties of the record Door can be modified externally from the user, but it should be set as a private record, so it can only be modified with the provided setters, having predefined behaviors.