Coursework - BorstMobile

Jake Burton 40278490

SET10112

Abstract. In this paper I will describe the design and implementation of a controller system for a nuclear-powered train known as the Borst Atomic Train. The introduction will contain an overview of the problem and developed solution. This will be followed by a higher level description of the solution with explanation on why it has been developed in that way, followed by descriptions of the individual procedures and functions found in the project code. Finalising the main body of this report will be formal verification of several procedures and a conclusion detailing the shortcomings of the developed solution.

1 Introduction

The Borst Atomic train was a proposed locomotive design powered by nuclear energy that was never fully developed as the system was deemed too expensive at the time of its proposal, 1954. As the design had various subsystems that dealt with dangerous hazards such as radiation, a control system needed developed. To provide a modern solution to this problem, I have developed a prototype version of the control system using ADA Spark.
The system was developed using the following rules that were provided by the specification:

1. The generator uses 5 control rods that limit the heat that is generated, however, there must always be at least 1 control rod in the reactor. The higher the heat in the reactor, the more electricity/power is produced,
2. The reactor has a water supply that provides further cooling,
3. The train must come to an emergency stop if the reactor overheats,
4. The reactor must be able to switch off to safely allow for maintenance,
5. The train's carriages can only be added or removed one at a time and only when the train is not moving,
6. The speed of the train will depend on the power generated fro the reactor and the number of carriages that are attached,
7. The train must have an absolute limit to it's speed.

While these rules listed cover most of the general functions of the train, the developed system was an extension for the carriages subsystem including calculations for the current speed of the train based on the weight of the carriages which, in turn, depends on the type of train, the number of passengers/tonnes of cargo and the number of attached cars. Furthermore, a series of warning lights have been added to the system that indicate the level of water in the tank, the heat of the engine and the state of the engine for maintenance purposes.

2 Controller Structure

To accommodate for the extensions to the system from the specification, a further set of rules has been implemented. These rules are:

1. The Maximum safe operating temperature for the generator is $2200°C$. Anything above this temperature will cause the engine to emergency stop, inserting all control rods and turning the engine off,
2. A maximum of 10 Carriages may be added to the Train that carry Passengers OR Freight but not both,
3. Engine Damage is increased with every action,
4. The Train can only be started if the level of water in the tank is above 30% full, all control rods are inserted and the Train's Generator is not too damaged,
5. Freight/Passengers may be added or removed one at a time when the train is not moving and the carriages are of the matching type, *i.e. Passengers cannot enter a Freight Train,*
6. The Carriage type can only be changed when all of the carriages are empty and the Train is not moving.

Based on these rules, I have created a series of unique data types that cover all of the variables and I have sorted these into three different record types: Generator, Carriage and Train. I have organised the data in this way as I feel that it makes the most sense to keep all related sets of information in the same place. Furthermore, the value ranges for the different data types are taken as approximate representations of values as found in real life *i.e. Train weight, Heat produced by the generator etc.* Furthermore, the heat produced was found to be analogous with the electricity produces, therefore, only heat is used.

2.1 Engine Record

The Engine record covers all variables relevant to power generation and indicators for the health of the Train's generator.

Engine			
Variable	**Type**	**Values**	**Function**
Rods	Control Rod	(1..5)	Number of Rods in reactor
Powered	Boolean	(On, Off)	Engine State
WaterTank	Level	(0..10)	Provides the level for the water tank in tonnes
Heat	EntineTemp	(0..2500)	Denotes the temperature of the generator
Maintenance	LastMaintained	(0..100)	Functions as damage meter
MtnLight	Light	(GREEN, AMBER, RED)	Indicates Maintenance Level
HeatLight			Indicates Heat in the Generator
WaterLight			Indicates the Water Level

The water is measured in tonnes to make the calculations for the total mass

easier later on. Furthermore, the Heat variable is an assumptions as I do not know the operating temperature of a nuclear reactor.

2.2 Carriage Record

The Carriage information consists of a variant record, with variants corresponding to the Passenger and Freight carriage types listed in the extended rules. When the Carriage variable is initialized, it is assumed to be a Freight train unless explicitly initialised as a Passenger train.

Carriage			
Variable	**Type**	**Values**	**Function**
Cars	NumCarriages	(0..10)	Serves as the number of carriages
MaxSpeed	Speed	(0..250)	Acts as the speed limit for the train based on the number of carriages
Weight	Mass	(190..1700)	Total weight of the train
Cargo	FreightWeight	(0..1100)	**VARIANT: Only for Freight Carriages**
Voyagers	PassengerCount	(0..1000)	**VARIANT: Only for Passenger Carriages**

The Cars variable is based on the value from the extended rules, and the Max Speed variable extends the maximum speed limit from the original set of rules from the specification. The Cargo variable is assumed to be measured in tonnes and makes the assumption that each Freight car itself weights 40 tonnes (this is relevant during weight calculations) and can carry 110 tonnes of cargo. Passenger cars are assumed to weigh 60 tonnes with a maximum of 100 passengers per car. Each passenger is assumed to be 70 kilograms based on my current weight.

2.3 Train Record

The Train record holds all of the information related to the train, including its generator and Carriages.

TRAIN			
Variable	**Type**	**Values**	**Function**
TrainType	CarType	(Freight, Passenger)	Denotes what the train is carrying
Cart	Carriage		Holds all of the Carriage information (BASED ON TrainType)
Generator	Engine		Holds all of the information related to the Train's Generator
CurSpeed	Speed	(0..250)	Acts as the current speed of the train

3 Descriptions of procedures and functions

This chapter aims to provide a more detailed look at the procedures and functions found within the program. The train record has been initialised in the

specification file with the name 'BorstMobile', meaning any mention of this name refers to the Train system. The BorstMobile can be assumed to be an In_Out variable for all procedures listed in this chapter. The SPARK level for each procedure will be included in the relevant subheading.

3.1 AddRod Procedure (GOLD)

This procedure adds 1 unit to the number of control rods and to the maintenance counter. The Generator Heat and Current Speed values require the calling of the functions GetHeat and GetCurrentSpeed. The pre-conditions for this are that the Train must be powered on, and range checks on the values that are changed. The post-condition is that the Train is still powered and that the number of Rods in the generator is greater than it's previous value.

3.2 RemoveRod Procedure (GOLD)

This procedure removes 1 unit to the number of control rods and adds 1 unit to the maintenance counter. The Generator Heat and Current Speed values require the calling of the functions GetHeat and GetCurrentSpeed. The pre-conditions for this are that the Train must be powered on, and range checks on the values that are changed. The post-condition is that the Train is still powered and that the number of Rods in the generator is less than it's previous value.

3.3 DoMaintenance Procedure (GOLD)

This procedure sets the maintenance counter to the minimum value, fills the water tank to its maximum value, sets the maintenance light and water light to GREEN and then calculates the new weight for the train as the weight of the water tank influences this. The Preconditions for this procedure are that the engine must be powered off with all control rods in the reactor. Additionally, the engine must not be moving and have the minimum heat value possible. The postcondition states that the engine must still be powered off, the maintenance counter must be equal to the minimum value possible and the water tank level must be at its maximum possible level.

3.4 StartEngine Procedure (GOLD)

Powers on the Train, resets the Generator's heat light to GREEN and calls the CalcWeight function to set the initial weight of the train. The preconditions are that the Train must not be powered and the Train must be travelling at the minimum speed and all control rods are inserted into the reactor. The only postcondition is that the Train must now be powered on.

3.5 StopEngine Procedure (GOLD)

This procedure powers off the Train, adds all of the control rods to the reactor, sets the trains speed and generator heat to the minimum values. The only precondition for this procedure is that the Train is powered and the postcondition states that the Train must no longer be powered.

3.6 AddTrainCar Procedure (GOLD)

This procedure adds a unit to the number of carriages in the carriage record, and sets the values for the maximum speed and total weight of the train by calling the GetMax and CalcWeight functions. The preconditions are that the number of cars already attached is less than the maximum and that the train currently is travelling at its minimum speed. The postcondition states the new number of cars must be smaller than or equal to the maximum number of cars.

3.7 RemoveTrainCar Procedure (GOLD)

This procedure removes a unit to the number of carriages in the carriage record, and sets the values for the maximum speed and total weight of the train by calling the GetMax and CalcWeight functions. The preconditions are that the number of cars already attached is more than the minimum and that the train currently is travelling at its minimum speed. The postcondition states the new number of cars must be greater than or equal to the minimum number of cars.

3.8 GetMax Function (GOLD)

This function returns the speed limit of the Train based on the number of carriages attached. The limit is equal to $200 - (n * 10)$ with n representing the number of carriages. There are only hidden preconditions that limit the range of n according to the NumCarriages variable type. The post condition states that the result speed returned by the function is less than or equal to 200km/h and greater than or equal to 100km/h

3.9 GetHeat Function (GOLD)

This function returns the heat in the Trains generator based on the number of control rods inserted and the level of water in the water tank. The returned value is equal to $((11-n)*((5-m)*55))$ where n is the level of water in the tank and m is the number of rods in the reactor. The post condition requires the result to be below or equal to the maximum value for engine temperature.

3.10 GetCurrentSpeed Function (GOLD)

This function returns the value for the current speed of the Train based on the heat from the generator and the information from the carriage record. The returned value is equal to $((n * 250)/2500) * (1 - (m/(1700 * 2)))$ where n is the temperature of the generator and m is the mass of the Train. The other numbers represent the maximum values from the Speed, Heat and Mass data types. Furthermore this calculation means that if the train is weighs 50% of the maximum Mass, then the train will be 25% slower. The postconditions only make sure that the value that is produced is within the limits of the Speed variable type.

3.11 EmergencyStop Procedure (GOLD)

This procedure stops and de-powers the Train, inserts all control rods into the reactor and sets the Generator heat light to RED if the Heat gets above $2200°C$. The only precondition for this procedure is that the Train is powered because the procedure could be called at any time.

3.12 FillTank Procedure (GOLD)

Fills the water tank to its maximum level, sets the Water Tank Light to GREEN and calls the CalcWeight function to set the weight. The precondition for this is that the Train is not moving, and the post condition states that the water tank must be full.

3.13 HeatL Procedure (GOLD)

This procedure changes the colour of the Generator's Heat light depending on the heat value. The precondition for this procedure is that the Train is powered and the postcondition is that the Generator's Heat light is GREEN, AMBER or RED. RED has been added as this procedure may still be called when the Train has had an Emergency Stop where the light value is changed to RED.

3.14 TankL Procedure (GOLD)

This procedure removes 1 unit of water from the water tank and adjusts the values for Weight, Heat and the Current Speed of the Train based on the new water value. Additionally, this procedure changes the colour of the Water Light depending on the Water value. The main precondition for this procedure is that the Train is powered and the postcondition is that the new value for $WaterTank$ is equal to $WaterTank'Old - 1$.

3.15 MaintL Procedure (GOLD)

This procedure changes the value of MtnLight based on the current maintenance counter value and stops the train if max value is reached. Preconditions for this are that the Train is powered and the maintenance is less than maximum.

3.16 SpeedL Procedure (GOLD)

Compares the values of current speed and max speed and adds rods to the reactor if current speed is higher. Only precondition is that the Train is powered.

3.17 ChangeCarriageType Procedure (GOLD)

Changes the type of train and carriage if no cars are attached to the opposite type.

3.18 AddPassenger/AddFreight Procedures (BRONZE?)

Adds a passenger/cargo to the train if train is not moving and train is PASSENGER/FREIGHT type and the train has enough carriages to accommodate the new weight. Unsure why they do not pass GOLD spark level. This is further examined in the next chapter.

3.19 RemovePassenger/RemoveFreight Procedures (BRONZE?)

Removes a passenger/cargo if the train is not moving and the train is PASSENGER/FREIGHT type. Unsure why they do not pass GOLD spark level.

3.20 CalcWeight Function (GOLD)

Returns the total weight of the train based on the level of the WaterTank, Units of Cargo/Passengers, TrainType and number of Carriages. Each person weighs 70kg, and PASSENGER train cars weigh 60 tonnes, whereas FREIGHT train cars weigh 40 tonnes. Postcondition states that the value must be below the maximum Mass.

4 Proof of Consistency

As the AddPassenger and AddFreight procedures serve functionally identical purposed for their respective TrainTypes, it makes sense to formally prove the contents of those procedures. As the main function of these methods it to add 1 unit to Voyagers/Cargo, only the line of code resposible for updating this number has been included in the following proof (I apologise for the strange layout, I do not know how to centre the proof on the page):

NC = Number of Carriages (N = NC'First, C = NC'Last)
PT = Passenger Total (P = PT'First, T = PT'Last)
Premises:
$n \in NC, n > N, n \leq C, v \in PT, p < T, p < T * n/C$
Conclusions:
$v + 1 \leq T, v + 1 \leq T * n/C, v + 1 \in PT$

$\Rightarrow$ *Premises* $\supset$ *Conclusions*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma...p < T \Rightarrow p+1 \leq T}\,Ax \quad \dfrac{\dfrac{}{\Gamma...p \in PT \Rightarrow p+1 \leq T}\,Ax \quad \dfrac{}{\Gamma...p < Tn/C \Rightarrow p+1 \leq Tn/C}\,Ax}{\Gamma \Rightarrow p+1 \in PT \wedge p+1 \leq Tn/C}\,R\wedge}{n \in NC, p \in PT, n > N, p < Tn/C, n \leq C, p < T \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,R\wedge}{n \in NC, p \in PT, n > N, p < Tn/C, n \leq C, p < T \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,L\wedge}{n \in NC, p \in PT, n > N, p < Tn/C, (n \leq C \wedge p < T) \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,L\wedge}{n \in NC, p \in PT, n > N, (p < Tn/C \wedge n \leq C \wedge p < T) \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,L\wedge}{n \in NC, p \in PT, (n > N \wedge p < Tn/C \wedge n \leq C \wedge p < T) \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,L\wedge}{n \in NC, (p \in PT \supset n > N \wedge p < Tn/C \wedge n \leq C \wedge p < T) \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,L\wedge}{(n \in NC \wedge p \in PT \supset n > N \wedge p < Tn/C \wedge n \leq C \wedge p < T) \Rightarrow (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,L\wedge}{\Rightarrow (n \in NC \wedge p \in PT \supset n > N \wedge p < Tn/C \wedge n \leq C \wedge p < T) \supset (p+1 \leq T \wedge p+1 \in PT \wedge p+1 \leq Tn/C)}\,R\supset$$

This proof shows that the existing preconditions and postconditions should avoid any runtime errors concerned with the addition of Voyagers/Cargo. However, the procedure also includes a line that calls the CalcWeight function, meaning that the existing pre- and postconditions for this procedure should be extended to include the necessary conditions for the CalcWeight function in order to fully avoid runtime errors. Unfortunately I do not know how to formally prove that function, so I have had to guess the correct pre- and postconditions. Furthermore, the rules mentioned previously mean that the Train must not be moving for weight to be added or removed.

5 Conclusion

While the majority of the control system passes GPS's SPARK gold level checks, not all procedures pass to this level, and as such, I believe that the control system should be considered as at a bronze level. This is one of the major shortcomings of this system. Furthermore, I have made the system massively overcomplicated and added many unnecessary features that are not needed for the full function of a control system, for example, I believe that it is possible to mostly avoid using record type variables. Due to this, if I were to redesign the system from scratch, I would either remove or streamline the way the records are organised. For example, the different components of the train could be organised using file inheritance.

Additionally, while some concurrency has been implemented in the main file to the ADA scripts, I would like to further develop the procedures so that they gradually increase the heat of the Generator and the speed of the Train. Currently, the main file only implements the procedures that deal with the Heat light, Water tank and light, the speed limit of the Train and the emergency stop with any concurrency. This could be further improved with a user interface that displays the different information for the train in real-time.