**D2 Submission Report**

**Class Diagram**

The class diagram shows the structure of each of the classes present in BusInfo as well as the relationships between them. BusInfo contains a number of non primitive data types that are detailed in the class diagram.

The Date datatype will contains 3 UnlimitedNatural numbers responsible for storing the year, month and day. The Date data type has an invariant listed which ensures the date is a valid one based on number of days in that month, 12 being the maximum month and the days in February based on if it’s a leap year or not. There is no primitive data type for storing dates so this was necessary. The next abstract data type is the the Time, this consists of 3 UnlimitedNatural numbers for the hours, minutes and seconds as well as a date field of the Date data type for storing the date on which that time is present. The time has a simple invariant where the hours cannot exceed 24, minutes 60 and seconds 60. Direction is an Enumerated data type that has a possible value of “Forwards” and “Backwards” used to indicate if the direction in which the journey is going. Timetable, BusID and RouteID are simple data types for storing their respective data.

The BusInfo class is the only class which has operations in it. It has ownership of most of the other classes as it will need to access them to carry out the functions, the only exception to this is the BusStop class which is owned by Route and does not need to be accessed via a function. This class will be used for all the administrative operations carried out by human controllers. BusInfo has a one way relationship with all it’s linked classes as it will need to access them all but they will not need to access the BusInfo class. BusInfo has a one to one or more relationship with every linked class as there will be many of each object in the system.

The Bus class models buses present in the system and contains all the variables that identify a bus and store its current state. The bus has a one way, one to 0 or more relationship with the bus route it’s on as well as the journey it's doing because 1 bus will only ever have no route or journey or have 1. The bus has a two way association with the driver as the driver would need to know the state of bus object as well as the bus object knowing it’s driver.

The Journey object models a single journey running on a route and only stores the ID, direction, bus and route the journey is happening on. The journey has a bidirectional association with both the bus and route as both of those classes would need to access it’s information. Journey has a one or more to one relationship with the route as each journey has one route and a route can have many journeys.

The route stores the Id, routeNumber (route name) and timetable as well as the bus stops and journeys on that route. The route has a unidirectional association with BusStop, that being one Route to one or more BusStops. The route has a unidirectional relationship with bus where Route doesn’t need to access the buses on it.

BusStop models a bus stop and only stores the ID of the stop and it’s address.

**Sequence Diagram**

The sequence diagram is a relatively simple one due to every function in BusInfo being a single message from the human controller to the BusInfo system. Most functions send a single message with no reply from the system as they are changing something within BusInfo. ViewJourneysOnRoute(routeID) has the HC send a routeID object message to the system and a Journey[0..\*] object is returned with all the journeys on that route.

ViewRouteTimetable(routeNumber) takes a route number string which it sends and a Timetable object of that route is returned in a reply.

GetBusState(busID) takes an integer which it sends and a Bus object of that ID is returned back to the HC.

All other functions are modifying the state of some parts of the system and thus do not receive a reply back.

**Contracts & Snapshots**

Snapshots SS-1 - SS-4 show three valid and one invalid state of the system. SS-1 and SS-4 show more complex states of the system involving many drivers, routes and busses. In each snapshot you can see the BusInfo class containing an array of all buses, drivers, journeys and routes on the system. This demonstrates the ownership shown in the UML models composite associations with most objects on the system. Due to this BusInfo is able to access all the objects in the system which it stores in those arrays.

SS-3 is a valid negative state because it shows two busses linked to a single journey via the DoesJourney association. This is wrong as the class diagram shows that one bus can only be linked to 0 or one journey.

SP-1 demonstrates the addNewBus function whose actions are detailed in the first contract. This function relies on the new bus not being already present in the system and there being a bus needing to be added. The SS diagram shows the function creating a B2, a new bus object from all the passed parameters that was not present before. SP-2 is a negative state as the function created a Bus with the same bus ID as B1, the contract states the new ID must be unique so this is wrong.

SP-3 shows modifyBus working correctly which relies on the bus the bus being present on the system as well as the bus ID passed being correct and the bus not doing a journey. SP-3 shows the right bus being correctly modified. SP-4 is a negative state as instead of modifying the existing bus, a new bus object with the new parameters is created with the ID passed. The contract doesn’t state a new bus gets created so this is wrong.

SP-5 shows viewJourneysOnRoute working correctly due to the pre-conditions being met. As the contract states, a new Journey array is created from the journeys present in BusInfo and this array gets returned. SP-6 is negative as it shows an empty array being created and returned despite journeys being present.

SP-7 shows viewRouteTimetable working, it relies on the route passed being present in the system and having a timetable. The timetable object is gotten from the route and is returned. SP-8 is a negative state as it creates and returns a null timetable object instead of the correct one.

SP-9 shows viewRouteTimetable which relies on the bus passed being present in the system. The bus object is found by ID and returned. SP-10 is negative as a bus with the wrong ID is returned.

SP-11 shows cancelJourney which relies on the journey existing, currently running and having a bus assigned. This is a positive state as the journey with ID 2 is assigned null, boolean doingJoruney of it’s bus is set to false and the journey is removed from the Journey[] array in BusInfo. SP-12 is negative as it shows the journey remains the same and present in BusInfo’s array and doingJourney is unchanged.

SP-13 shows assignDriverToBus, it relies on both the driver and bus IDs passed having a corresponding object, the bus being in service and not doing a journey as well as the driver not driving another bus. This state is positive as the driver is now associated with the bus passed. SP-14 is negative as it allows for the driver being assigned to an already occupied bus which is forbidden in the contract.

SP-15 shows addNewDriver which relies on the driver not being in the system already and needing to be added. The state is positive as a new driver object is created with the passed parameters. SP-16 is negative as the contract states the ID generated is unique whereas the state shows a colliding ID.

SP-17 shows setInOrOutOfService which relies on the bus existing and not doing a journey. The state is positive as the bus is assigned the passed boolean. SP-18 is negative as it fails to change the inService boolean of the bus.

SP-19 shows assignBusToRoute which relies on the bus not doing a journey and it and the route being valid. The state is positive as a new HasAssignedRoute association is made. SP-20 is negative as the association is made with a bus of the wrong ID which is not how it’s supposed to work based on the contract.