Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Rome2Rio

MIEIC05:

Rui Pedro Machado Araújo - up201403263@fe.up.pt

António Miguel Silva Pereira - up201307910@fe.up.pt

FEUP - MIEIC, 2019

Index

1. Informal system description and list of requirements

1.1 Informal system description

This is a program that depicts the “Rome2Rio” website where a user can insert it’s starting and destination points and receive the possible routes by train, bus, ferry, car or plane allowing the user to choose the one best fitting for him in regards of duration, transportation type and cost.

In addition to this, we also simulated the possibility of existing a platform admin that manages routes in the system.

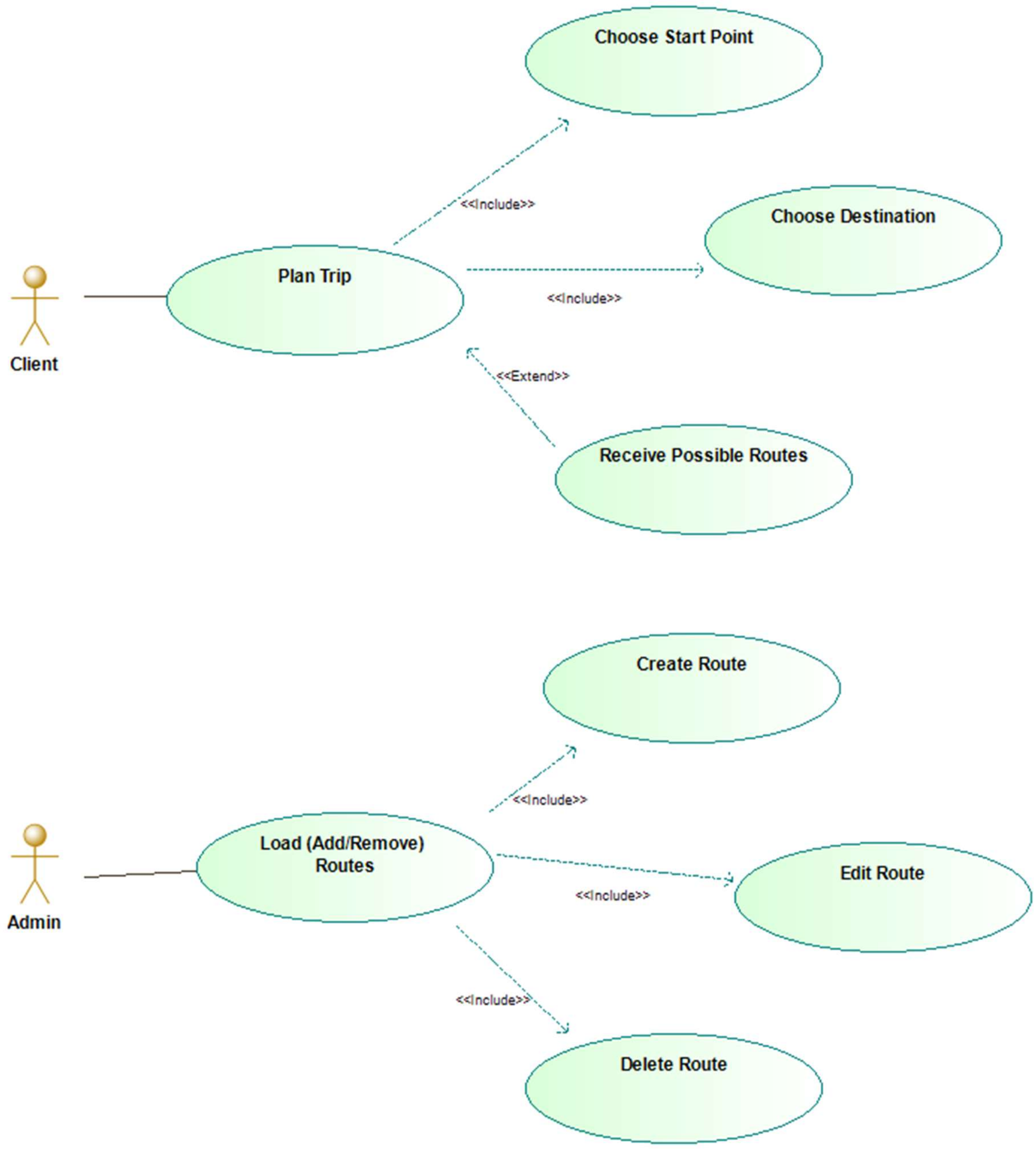
1.2 List of requirements

Id	Priority	Description
R1	Mandatory	The Client must be able to choose its starting point. Given that it exists.
R2	Mandatory	The Client must be able to choose the destination point. Given that it exists.
R3	Mandatory	The program should return to the client all the possible routes from starting point to the destination. If there is a connection between the two places.
R4	Mandatory	The Admin must be able to create and delete routes.
R5	Opcional	The Admin must be able to edit routes.

These requirements are directly translated onto use cases as shown next.

2. Visual UML model

2.1 Use case model

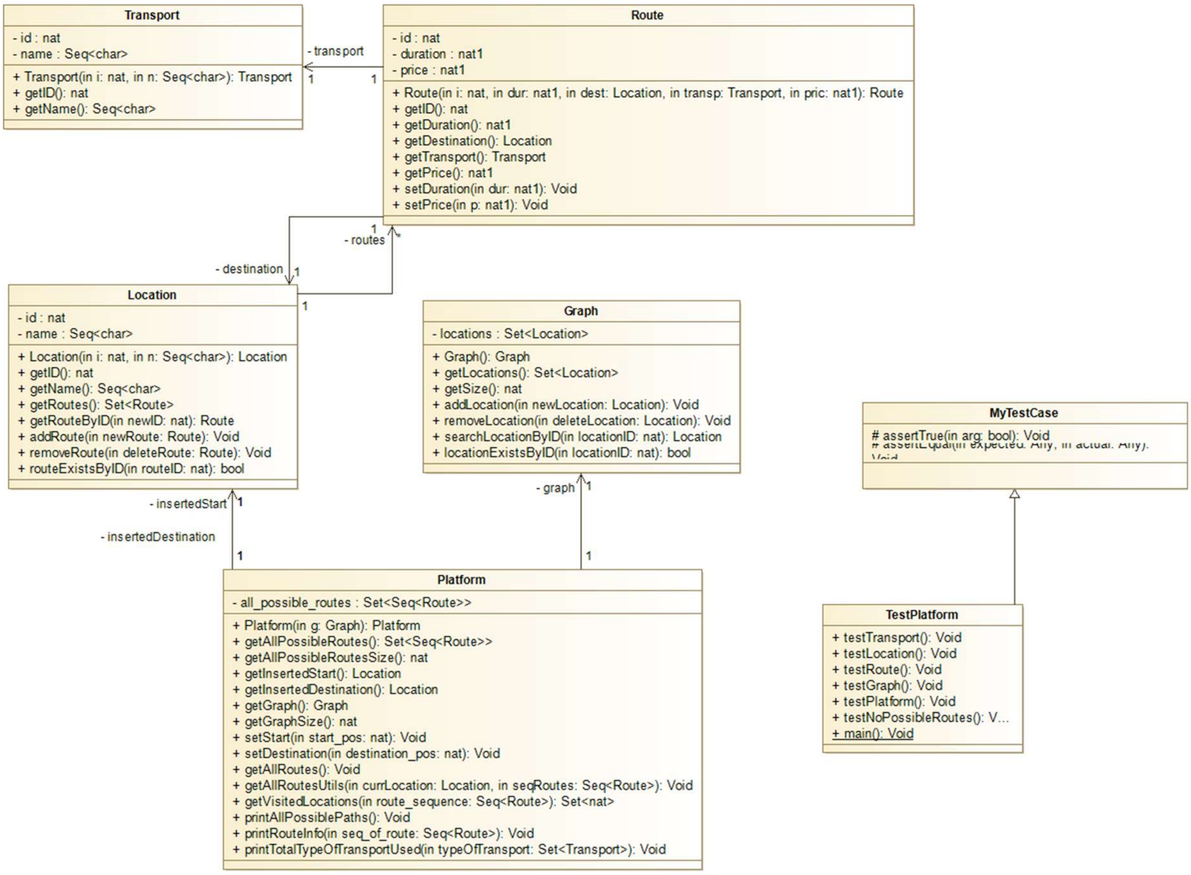


The major use case scenarios (to be used later as test scenarios) are described next.

Scenario	**Plan Trip**
Description	Normal trip planning in the platform.
Pre-conditions	1. The platform is online awaiting requests. (initial system state) 2. Start Point and Destination locations must exist in the platform.
Post-conditions	1. The customer receives all possible routes from the starting location to the destination. If there is no connection the customer is warned. 2. The platform is awaiting requests again. (final system state)
Steps	1. The customer inserts a starting point. 2. The customer inserts a destination. 3. The platform displays all the possible routes between them.
Exceptions	1. When inserting a starting and destination point the platform verifies if it exists in the Graph.

Scenario	**Load Routes**
Description	Normal scenario for add/edit/remove routes from the platform.
Pre-conditions	*1.* The platform is online awaiting requests. *2.* Each route is unique(ID). 3. If adding a route its start and destination point along with its transport type must exist in the platform.
Post-conditions	1. The route was correctly added/removed to the location. 2. The route was edited with the correct values.
Steps	(unspecified)
Exceptions	(unspecified)

2.2 Class Model


Transport
- id : nat
- name : Seq<char>
+ Transport(in i: nat, in n: Seq<char>): Transport
+ getID(): nat
+ getName(): Seq<char>
- transport
1
1
Route
- id : nat
- duration : nat1
- price : nat1
+ Route(in i: nat, in dur: nat1, in dest: Location, in transp: Transport, in pric: nat1): Route
+ getID(): nat
+ getDuration(): nat1
+ getDestination(): Location
+ getTransport(): Transport
+ getPrice(): nat1
+ setDuration(in dur: nat1): Void
+ setPrice(in p: nat1): Void
1
- routes
*
- destination
1
1
Location
- id : nat
- name : Seq<char>
+ Location(in i: nat, in n: Seq<char>): Location
+ getID(): nat
+ getName(): Seq<char>
+ getRoutes(): Set<Route>
+ getRouteByID(in newID: nat): Route
+ addRoute(in newRoute: Route): Void
+ removeRoute(in deleteRoute: Route): Void
+ routeExistsByID(in routeID: nat): bool
Graph
- locations : Set<Location>
+ Graph(): Graph
+ getLocations(): Set<Location>
+ getSize(): nat
+ addLocation(in newLocation: Location): Void
+ removeLocation(in deleteLocation: Location): Void
+ searchLocationByID(in locationID: nat): Location
+ locationExistsByID(in locationID: nat): bool
MyTestCase
# assertTrue(in arg: bool): Void
# assertEqual(in expected: Any, in actual: Any): Void
- insertedStart
1
- insertedDestination
1
- graph
1
1
Platform
- all_possible_routes : Set<Seq<Route>>
+ Platform(in g: Graph): Platform
+ getAllPossibleRoutes(): Set<Seq<Route>>
+ getAllPossibleRoutesSize(): nat
+ getInsertedStart(): Location
+ getInsertedDestination(): Location
+ getGraph(): Graph
+ getGraphSize(): nat
+ setStart(in start_pos: nat): Void
+ setDestination(in destination_pos: nat): Void
+ getAllRoutes(): Void
+ getAllRoutesUtils(in currLocation: Location, in seqRoutes: Seq<Route>): Void
+ getVisitedLocations(in route_sequence: Seq<Route>): Set<nat>
+ printAllPossiblePaths(): Void
+ printRouteInfo(in seq_of_route: Seq<Route>): Void
+ printTotalTypeOfTransportUsed(in typeOfTransport: Set<Transport>): Void
TestPlatform
+ testTransport(): Void
+ testLocation(): Void
+ testRoute(): Void
+ testGraph(): Void
+ testPlatform(): Void
+ testNoPossibleRoutes(): V...
+ main(): Void

Class	Description
Transport	Defines a mean of transport to be used in a route between two locations
Route	Defines a connection between two locations with a price, duration time and a mean of transport
Location	Defines a Place in a map from where a customer can start or travel to.
Graph	Defines a set of interconnected locations from where a

	customer can travel to.
Platform	Defines the online platform where the user searches for possible routes from a starting location to the destination.
MyTestCase	Superclass for test classes; defines assertEquals and assertTrue.
Test Platform	Defines the test/usage scenarios and test cases for the platform.

3. Formal VDM++ model

3.1 Class Transport

```
class Transport
/*
      Class that represents a type of transportation such as Plane, Ferry Bus And
Train or more.
*/
instance variables
  private id:nat;
  private name:seq1 of (char);

operations

      /* Constructor Transport - id and name with length > 0 */
  public  Transport : nat * seq1 of (char) ==> Transport
  Transport(i, n) == (
          id := i;
          name := n;
      return self
    );

  /* ------------------ Getters ------------------ */

  /* Returns the transport ID */
  public getID : () ==> nat
  getID() == (
      return id;
  );
  /* Returns the transport name */
  public getName : () ==> seq1 of (char)
  getName() == (
       return name;
  );

end Transport
```

3.2 Class Location

```
class Location
/*
       This class represents a location of a city with id and name,
       it ilustrates the state of Node with Edges in a Graph environment.
*/

instance variables
  private id:nat;
  private name:seq1 of (char);
  private routes:set of (Route) := {}; /* Edges of a Node(Location) */

operations

       /* Constructor Location - with id, name and empty set of routes*/
  public Location : nat * seq1 of (char) ==> Location
  Location(i, n) == (
           id := i;
           name := n;
           return self
    )
    post routes = {};


  /* ------------------ Getters ------------------ */

  /* Returns the location ID */
  pure public getID : () ==> nat
  getID() == (
       return id;
  );

  /* Returns the location name */
  public getName : () ==> seq1 of (char)
  getName() == (
       return name;
  );

  /* Returns a set of Routes(Edges) */
  public getRoutes : () ==> set of (Route)
  getRoutes() == (
       return routes;
  );

  /* Returns a Route with a given ID, if non-existent returns a empty Route */
  public getRouteByID : nat ==> Route
  getRouteByID(newID) == (
             dcl tempRoute : Route;
      for all route in set routes do (
             if route.getID() = newID
             then tempRoute := route;
      );
      return tempRoute;
  );


      /* ------------------ Methods ------------------ */

  /* Adds a Route(Edge) to a Location(Node) - Admin Only */
  public addRoute : Route ==> ()
  addRoute(newRoute) == (
      routes := {newRoute} union routes
  )
  pre not newRoute in set routes
  post newRoute in set routes;

  /* Removes a Route(Edge) from a Location(Node) - Admin Only */
  public removeRoute : Route ==> ()
  removeRoute(deleteRoute) == (
      routes  := routes \ {deleteRoute}
  )
  pre deleteRoute in set routes
  post not deleteRoute in set routes;

  /*
        Verifies if a route with a given ID exists
        If it exists return true and false otherwise
  */
  public routeExistsByID : nat ==> bool
  routeExistsByID(routeID) == (
      for all route in set routes do (
             if route.getID() = routeID
             then return true;
      );
      return false;
  );

end Location
```

3.3 Class Route

```
class Route
/*
      Class that represents a connection between two locations
      stating several information such as id, duration of the trip, destination
of the trip
      transport type of trip and the cost of the trip.
*/

instance variables
  private id:nat;
  private duration:nat1;
  private destination:Location;
  private transport:Transport;
  private price:nat1;

operations

       /* Constructor Route - with id, duration of the trip, destination, 
transport type and price */
  public Route : nat * nat1 * Location * Transport * nat1 ==> Route
  Route(i, dur, dest, transp, pric) == (
           id := i;
           duration := dur;
           destination := dest;
           transport := transp;
           price := pric;
       return self
    );

       /* ------------------ Getters ------------------ */

        /* Returns the Route ID */
  public getID : () ==> nat
  getID() == (
       return id;
  );

   /* Returns the Route duration */
  public getDuration : () ==> nat1
  getDuration() == (
       return duration;
  );

   /* Returns the Route destination */
  public getDestination : () ==> Location
  getDestination() == (
       return destination;
  );

   /* Returns the Route transport */
  public getTransport : () ==> Transport
  getTransport() == (
       return transport;
  );

   /* Returns the Route price */
  public getPrice : () ==> nat1
  getPrice() == (
       return price;
  );

  /* Sets the new duration - Admin Only */
  public setDuration : nat1 ==> ()
       setDuration(dur) == (
             duration := dur;
  );

  /* Sets the new price - Admin Only */
  public setPrice : nat1 ==> ()
```

```
             setPrice(p) == (
             price := p;
  );


end Route
```

3.4 Class Graph

```
class Graph
/*
      Class that holds a set of Locations that represent Nodes in the typical
Graph-Node-Edge representation.
*/
types
values
instance variables
  private locations:set of (Location); /* represents nodes in a typical graph */

operations

      /* Constructor Graph - with set locations that are Nodes - initially it is
empty*/
  public Graph : () ==> Graph
  Graph() == (
      locations := {};
        return self
  );

  /* ------------------ Getters ------------------ */

      /* Returns a set of Locations(Nodes) */
  public getLocations : () ==> set of (Location)
        getLocations() == (
             return locations;
  );

  /* Returns the number of nodes(Locations) in a graph */
  public getSize : () ==> nat
        getSize() == (
             return card locations;
  );

  /* Adds a location to the graph */
  public addLocation : Location ==> ()
        addLocation(newLocation) == (
             locations := {newLocation} union locations;
  )
  pre not newLocation in set locations
  post newLocation in set locations;

  /* Removes a Location from the graph */
  public removeLocation : Location ==> ()
        removeLocation(deleteLocation) == (
             locations := locations \ {deleteLocation};
  )
  pre deleteLocation in set locations
  post not deleteLocation in set locations;

  /* Returns a location given an ID - if non-exist returns empty location */
  public searchLocationByID : nat ==> Location
  searchLocationByID(locationID) == (
      dcl tempCity : Location;
      for all city in set locations do (
             if city.getID() = locationID
             then tempCity := city;
      );
      return tempCity;
  );

  /* Verifies if a location by the given ID exists
       return true if it does and false if it doesn't
  */
  public locationExistsByID : nat ==> bool
  locationExistsByID(locationID) == (
      for all city in set locations do (
             if city.getID() = locationID
             then return true;
      );
      return false;
  );

functions

traces

end Graph
```

3.5 Class Platform

```
class Platform
/*
      Class that represents the application Rome2Rio it holds
      the inserted start point and destination the user introduces,
      along with the Graph where all the Nodes(Locations) and also all the
possible routes existent
      (all possible ways to go from a pointA to pointB)
*/

instance variables
  private insertedStart:Location;
  private insertedDestination:Location;
  private graph:Graph;
  private all_possible_routes: set of seq of Route;

operations
  public Platform : Graph ==> Platform
        Platform(g) == (
             graph := g;
             all_possible_routes := {};
             return self
  );

  /* ------------------ Getters ------------------ */

  /* Returns all the possible routes given from the user */
  public getAllPossibleRoutes : () ==> set of seq of Route
             getAllPossibleRoutes() == (
             return all_possible_routes;
        );

  /* Returns all the possible routes size */
  public getAllPossibleRoutesSize : () ==> nat
             getAllPossibleRoutesSize() == (
             return card all_possible_routes
      );

  /* Returns the user inserted start desired point */
  public getInsertedStart : () ==> Location
  getInsertedStart() == (
      return insertedStart
  );

  /* Returns the user inserted destination point */
  public getInsertedDestination : () ==> Location
  getInsertedDestination() == (
      return insertedDestination
  );

  /* Returns the graph that holds all the locations of the platform */
  public getGraph : () ==> Graph
  getGraph() == (
      return graph
  );

  /* Returns the number of locations in the platform */
  public getGraphSize : () ==> nat
             getGraphSize() == (
                    return graph.getSize();
      );


  /* ------------------ Setters ------------------ */

      /* Sets a starting point for the user */
  public setStart : nat ==> ()
  setStart(start_pos) == (
      if graph.locationExistsByID(start_pos)
      then insertedStart := graph.searchLocationByID(start_pos)
  );

  /* Sets a destination point for the user */
  public setDestination : nat ==> ()
        setDestination(destination_pos) == (
          if graph.locationExistsByID(destination_pos)
             then insertedDestination := graph.searchLocationByID(destination_pos)
  );

  /*
```

```
      Function that calculates all the possible routes from pointA to pointB
      and saves them in all_possible_routes variable
  */
  public getAllRoutes : () ==> ()
  getAllRoutes() == (
      dcl routes : seq of Route := [];
      getAllRoutesUtils(insertedStart, routes);
      );

      /* Auxiliar function for getAllRoutes to parse through each seq of Route */
      public getAllRoutesUtils : Location * seq of Route ==> ()
      getAllRoutesUtils(currLocation, seqRoutes) == (
             for all route in set currLocation.getRoutes() do (
                   dcl newRoute: seq of Route := seqRoutes;
                   dcl routeDestinationID : nat :=
route.getDestination().getID();
                   if routeDestinationID not in set
getVisitedLocations(seqRoutes)
                   then (
                         newRoute(len newRoute + 1) := route
                   );

                   if route.getDestination() = insertedDestination
                   then (
                         all_possible_routes := all_possible_routes union
{newRoute};
                   )
                   else (
                         dcl numberOfEdgesInDestination: nat := card
route.getDestination().getRoutes();

                         if routeDestinationID not in set
getVisitedLocations(seqRoutes) and numberOfEdgesInDestination > 0
                         then (

      getAllRoutesUtils(route.getDestination(),newRoute)
                         );
                   );
             );
      );

      /* Returns a set of all the locations ID already visited */
      public getVisitedLocations: seq of Route ==> set of nat
      getVisitedLocations(route_sequence) == (
             dcl locationsID : set of nat := {insertedStart.getID()};
             for all route in set elems route_sequence do (
                   locationsID := {route.getDestination().getID()} union
locationsID;
             );
             return locationsID;
      );

      /* Function that prints the possible routes the user desires */
      public printAllPossiblePaths : () ==> ()
      printAllPossiblePaths() == (
             IO`print("\n");
             IO`print("Start: ");
             IO`println(insertedStart.getName());
```

```
        IO`print("Destination: ");
        IO`println(insertedDestination.getName());
        IO`print("-------------------- \n\n");
        if card all_possible_routes > 0
        then (
            dcl local_i : nat1 := 1;
            for all routes in set all_possible_routes do (
                IO`print("Route: ");
                IO`println(local_i);
                printRouteInfo(routes);
                local_i := local_i + 1;
                IO`print("\n");
            );
        )
        else (
            IO`print("Not Possible to reach the Destination from that
Starting Point!");
            IO`println("\n");
            );
    );
    /* function that prints the information of each individual route and also
the total costs of everything */
    public printRouteInfo : seq of Route ==> ()
    printRouteInfo(seq_of_route) == (
        dcl prevRoute : Route;
        dcl totalTripDuration : nat := 0;
        dcl totalTripCost : nat := 0;
        dcl totalTypeOfTransport : set of Transport := {};
        for i=1 to len seq_of_route do (
                if i=1
                    then (
                        IO`print("Travel from: ");
                        IO`print(insertedStart.getName());
                        IO`print(" to: ");

    IO`print(seq_of_route(i).getDestination().getName());
                        IO`print(" transport: ");

    IO`print(seq_of_route(i).getTransport().getName());
                        IO`print(" duration: ");
                        IO`print(seq_of_route(i).getDuration());
                        IO`print(" with Price: ");
                        IO`println(seq_of_route(i).getPrice());
                        totalTripDuration := totalTripDuration +
seq_of_route(i).getDuration();
                        totalTripCost := totalTripCost +
seq_of_route(i).getPrice();
                        totalTypeOfTransport :=
totalTypeOfTransport union {seq_of_route(i).getTransport()};
                    )
                    else (
                        IO`print("Travel from: ");

    IO`print(prevRoute.getDestination().getName());
                        IO`print(" to: ");

    IO`print(seq_of_route(i).getDestination().getName());
                        IO`print(" transport: ");
        IO`print(seq_of_route(i).getTransport().getName());
                                                IO`print(" duration: ");
                                                IO`print(seq_of_route(i).getDuration());
                                                IO`print(" with Price: ");
                                                IO`println(seq_of_route(i).getPrice());
                                                totalTripDuration := totalTripDuration +
seq_of_route(i).getDuration();
                                                totalTripCost := totalTripCost +
seq_of_route(i).getPrice();
                                                totalTypeOfTransport :=
totalTypeOfTransport union {seq_of_route(i).getTransport()};
                        );
                        prevRoute := seq_of_route(i); --used to save the previous
route location
                );
                IO`print("\n");
                IO`print("Total travel time: ");
                IO`print(totalTripDuration);
                IO`println(" hours.");
                IO`print("Total travel price: ");
                IO`print(totalTripCost);
                IO`println(" euros.");
                IO`print("Types of transport used: ");
                printTotalTypeOfTransportUsed(totalTypeOfTransport);
                IO`print("\n");
        );

        /* function to print the types of transports used in a trip */
        public printTotalTypeOfTransportUsed : set of Transport ==> ()
        printTotalTypeOfTransportUsed(typeOfTransport) == (
                dcl local_i : nat := 0;
                for all transport in set typeOfTransport do (
                        if local_i=0
                                then
                                        IO`print(transport.getName())
                                else (
                                        IO`print(" -- ");
                                        IO`print(transport.getName());
                                );
                        local_i := local_i + 1;
                );

        )

end Platform
```

4. Model Validation

4.1 Class MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
*/

operations

      -- Simulates assertion checking by reducing it to pre-condition checking.
      -- If 'arg' does not hold, a pre-condition violation will be signaled.
      protected assertTrue : bool ==> ()
      assertTrue(arg) ==
            return
      pre arg;

      -- Simulates assertion checking by reducing it to post-condition checking.
      -- If values are not equal, prints a message in the console and generates
      -- a post-conditions violation.
      protected assertEqual : ? * ? ==> ()
      assertEqual(expected, actual) ==
            if  expected <> actual then  (
                  IO`print("Actual value(");
                  IO`print(actual);
                  IO`print(") different from expected(");
                  IO`print(expected);
                  IO`println(")\n")
            )
            post expected = actual
end MyTestCase
```

4.2 Class TestPlatform

```
class TestPlatform is subclass of MyTestCase

instance variables

      --Graph
      graph : Graph := new Graph();
      graph2 : Graph := new Graph();

      --Platform
      platform : Platform := new Platform(graph);
      platform2 : Platform := new Platform(graph2);

      --Locations(ID,Name)
      location0 : Location := new Location(0,"Oporto");
      location1 : Location := new Location(1,"Lisbon");
      location2 : Location := new Location(2,"Faro");
```

```
    location3 : Location := new Location(3,"Madrid");
    location4 : Location := new Location(4,"Paris");
    location5 : Location := new Location(5,"London");
    location6 : Location := new Location(6,"Rome");
    location7 : Location := new Location(7,"Amsterdam");
    location8 : Location := new Location(8,"Berlin");
    location9 : Location := new Location(9,"Vienna");
    location10 : Location := new Location(10,"Barcelona");
    location100 : Location := new Location(100,"Tokyo");

    --Local Variables for testing
    test_location :     set of Location := {location0};
    test_all_possible : set of seq of Route := {};

    --Transports(ID,Name)
    transport0 : Transport := new Transport(0,"Car");
    transport1 : Transport := new Transport(1,"Train");
    transport2 : Transport := new Transport(2,"Plane");
    transport3 : Transport := new Transport(3,"Ferry");
    transport4 : Transport := new Transport(4,"Bus");
    transport100 : Transport := new Transport(100,"Space Shuttle");

    --Routes(ID,Duration,DestinationTransport,Price)
    route0 : Route := new Route(0,3,location1,transport0,20);
    route1 : Route := new Route(1,3,location3,transport1,30);
    route2 : Route := new Route(2,3,location3,transport2,40);
    route3 : Route := new Route(3,3,location3,transport3,50);
    route4 : Route := new Route(4,3,location4,transport4,60);
    route5 : Route := new Route(5,3,location0,transport0,70);
    route100 : Route := new Route(100,5,location0,transport100,100);

operations

    ------------- TRANSPORT TEST----------------------

    public testTransport : () ==> ()
    testTransport() == (
        assertEqual(transport100.getID(),100);
        assertEqual(transport100.getName(),"Space Shuttle")
    );

        ------------- LOCATION TEST----------------------

    public testLocation : () ==> ()
    testLocation() == (
        assertEqual(location100.getID(),100);
        assertEqual(location100.getName(),"Tokyo");
        assertEqual(card location100.getRoutes(),0);
        location100.addRoute(route100);
        assertEqual(card location100.getRoutes(),1);
        location100.removeRoute(route100);
        assertEqual(location100.routeExistsByID(100), false);
        assertEqual(card location100.getRoutes(),0)
    );

            ------------- ROUTE TEST----------------------

    public testRoute : () ==> ()
```

```
testRoute() == (
        assertEqual(route100.getID(),100);
        assertEqual(route100.getDestination(),location0);
        assertEqual(route100.getDuration(),5);
        assertEqual(route100.getTransport(),transport100);
        assertEqual(route100.getPrice(),100);
        route100.setPrice(90);
        route100.setDuration(3);
        assertEqual(route100.getPrice(),90);
        assertEqual(route100.getDuration(),3);
);

                    ------------- GRAPH TEST----------------------

public testGraph : () ==> ()
testGraph() == (
        assertEqual(graph.getSize(),0);
        graph.addLocation(location0);
        graph.addLocation(location1);
        assertEqual(graph.getSize(),2);
        assertTrue(graph.locationExistsByID(location1.getID()));
        assertEqual(graph.searchLocationByID(location1.getID()), location1);
        graph.removeLocation(location1);
        assertEqual(graph.locationExistsByID(location1.getID()),false);
        assertEqual(graph.getSize(),1);
        assertTrue(graph.getLocations() = test_location);
        graph.removeLocation(location0);
);

                          ------------- PLATFORM TEST----------------------

public testPlatform : () ==> ()
testPlatform() == (

        assertEqual(platform.getAllPossibleRoutesSize(), 0);
        assertEqual(platform.getAllPossibleRoutes(), {});
        assertEqual(platform.getGraph(),graph);
        assertEqual(platform.getGraphSize(),0);

        platform.getGraph().addLocation(location0);
        platform.getGraph().addLocation(location1);
        platform.getGraph().addLocation(location3);
        platform.setStart(0);
        platform.setDestination(3);
        assertEqual(platform.getInsertedStart(),location0);
        assertEqual(platform.getInsertedDestination(),location3);


platform.getGraph().searchLocationByID(location0.getID()).addRoute(route0);

platform.getGraph().searchLocationByID(location0.getID()).addRoute(route2);

platform.getGraph().searchLocationByID(location1.getID()).addRoute(route1);

assertTrue(platform.getGraph().searchLocationByID(location0.getID()).routeExistsByID(0));
```

```
        assertEqual(platform.getGraph().searchLocationByID(location0.getID()).getRo
uteByID(0),route0);
                assertEqual(card
platform.getGraph().searchLocationByID(location0.getID()).getRoutes(), 2);

                /* Test search algorithm */
                platform.getAllRoutes();
                platform.printAllPossiblePaths();

        );

        public testNoPossibleRoutes : () ==> ()
        testNoPossibleRoutes() == (
                platform2.getGraph().addLocation(location0);
                platform2.getGraph().addLocation(location1);
                platform2.getGraph().addLocation(location3);
                platform2.setStart(0);
                platform2.setDestination(3);

        platform2.getGraph().searchLocationByID(location0.getID()).addRoute(route0)
;

                platform2.getAllRoutes();
                platform2.printAllPossiblePaths();
        );

        public static main: () ==> ()
                main() == (
                        new TestPlatform().testTransport();
                        new TestPlatform().testLocation();
                        new TestPlatform().testRoute();
                        new TestPlatform().testGraph();
                        new TestPlatform().testPlatform();
                        new TestPlatform().testNoPossibleRoutes();

                );


traces
-- TODO Define Combinatorial Test Traces here
end TestPlatform
```

5. Model Verification

5.1 Example of operation establishes postcondition

One of the proof obligations generated by Overture is:

no	PO Name	Type
1	Graph`addLocation(Location)	operation establishes postcondition

The code under analysis (with the relevant part underlined) is:

```
public addLocation : Location ==> ()
        addLocation(newLocation) == (
              locations := {newLocation} union locations;
 )
 pre not newLocation in set locations
 post newLocation in set locations;
```

Proof obligation view:

(forall newLocation:Location & ((not (newLocation in set locations)) => (newLocation in set ({newLocation} union locations))))

In this case the proof is trivial because the postcondition is reassuring what was done in the body of the function. Checking if the union was correctly added.

5.2 Example of invariant verification

Non-existent due to no types in any class we have created.

5.3 Example of legal sequence application

no	PO Name	Type
8	Platforn`getAllRoutesUtils(Location,seq of (Route))	legal sequence application

```
/* Auxiliar function for getAllRoutes to parse through each seq of Route */
      public getAllRoutesUtils : Location * seq of Route ==> ()
      getAllRoutesUtils(currLocation, seqRoutes) == (
            for all route in set currLocation.getRoutes() do (
                  dcl newRoute: seq of Route := seqRoutes;
                  dcl routeDestinationID : nat :=
route.getDestination().getID();
                  if routeDestinationID not in set
getVisitedLocations(seqRoutes)
                  then (
                        newRoute(len newRoute + 1) := route
                  );

                  if route.getDestination() = insertedDestination
                  then (
                        all_possible_routes := all_possible_routes union
{newRoute};
                  )
                  else (
                        dcl numberOfEdgesInDestination: nat := card
route.getDestination().getRoutes();

                        if routeDestinationID not in set
getVisitedLocations(seqRoutes) and numberOfEdgesInDestination > 0
                        then (

      getAllRoutesUtils(route.getDestination(),newRoute)
                        );
                  );
            );
```

Proof obligation view generated by overture:

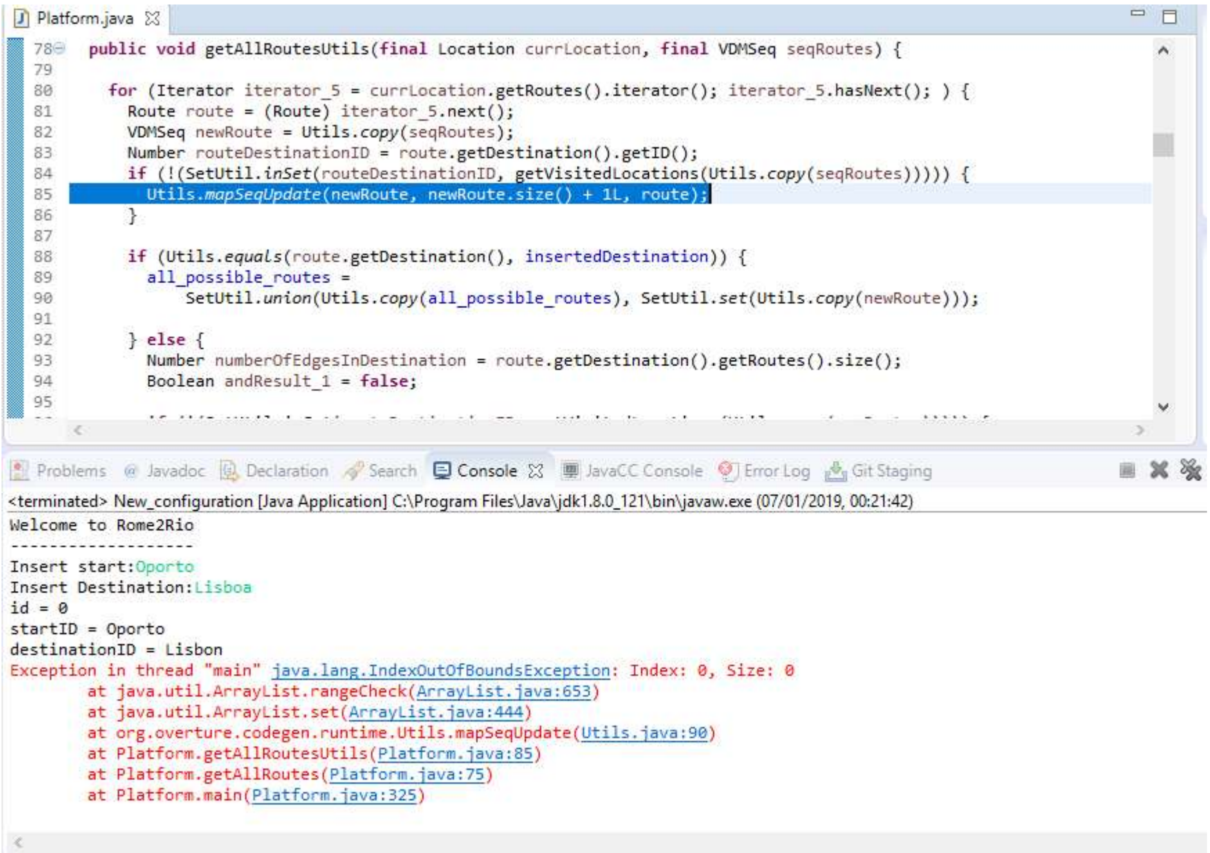
```
(forall currLocation:Location, seqRoutes:seq of (Route) & ((((len newRoute) + 1) >
0) and (((len newRoute) + 1) <= ((len newRoute) + 1))))
```

In this case the proof is trivial because the postcondition is reassuring what was done in the body of the function. Checking if the union was correctly added.

6. Code generation

Using the Overture code generation tool, we generated a java application to do further testing, including interface with a real customer. To do that we are using a simple console interface.

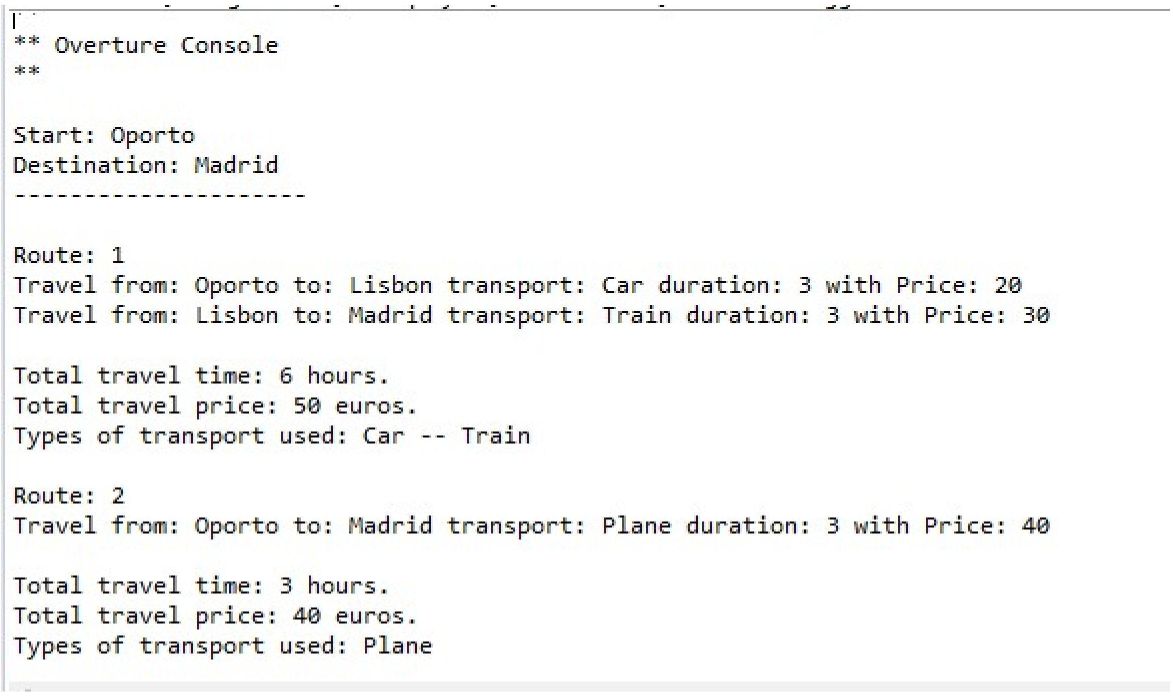
The user is expected to insert a starting point and a destination, and all the possible routes are shown (if both the Locations exist). Unfortunately, due to an error generating the java code we could not test it (see image below).



```java
public void getAllRoutesUtils(final Location currLocation, final VDMSeq seqRoutes) {

  for (Iterator iterator_5 = currLocation.getRoutes().iterator(); iterator_5.hasNext(); ) {
    Route route = (Route) iterator_5.next();
    VDMSeq newRoute = Utils.copy(seqRoutes);
    Number routeDestinationID = route.getDestination().getID();
    if (!(SetUtil.inSet(routeDestinationID, getVisitedLocations(Utils.copy(seqRoutes))))) {
      Utils.mapSeqUpdate(newRoute, newRoute.size() + 1L, route);
    }

    if (Utils.equals(route.getDestination(), insertedDestination)) {
      all_possible_routes =
          SetUtil.union(Utils.copy(all_possible_routes), SetUtil.set(Utils.copy(newRoute)));

    } else {
      Number numberOfEdgesInDestination = route.getDestination().getRoutes().size();
      Boolean andResult_1 = false;

```

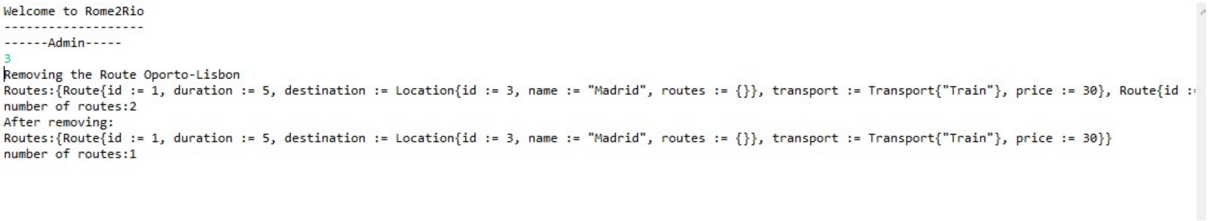
```
<terminated> New_configuration [Java Application] C:\Program Files\Java\jdk1.8.0_121\bin\javaw.exe (07/01/2019, 00:21:42)
Welcome to Rome2Rio
-------------------
Insert start:Oporto
Insert Destination:Lisboa
id = 0
startID = Oporto
destinationID = Lisbon
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
	at java.util.ArrayList.rangeCheck(ArrayList.java:653)
	at java.util.ArrayList.set(ArrayList.java:444)
	at org.overture.codegen.runtime.Utils.mapSeqUpdate(Utils.java:90)
	at Platform.getAllRoutesUtils(Platform.java:85)
	at Platform.getAllRoutes(Platform.java:75)
	at Platform.main(Platform.java:325)
```

The output should look like the one we got in the overture console during testing (show below):



```
** Overture Console
**

Start: Oporto
Destination: Madrid
---------------------

Route: 1
Travel from: Oporto to: Lisbon transport: Car duration: 3 with Price: 20
Travel from: Lisbon to: Madrid transport: Train duration: 3 with Price: 30

Total travel time: 6 hours.
Total travel price: 50 euros.
Types of transport used: Car -- Train

Route: 2
Travel from: Oporto to: Madrid transport: Plane duration: 3 with Price: 40

Total travel time: 3 hours.
Total travel price: 40 euros.
Types of transport used: Plane
```

As for the admin we were able to test everything successfully and below can be seen one example:



```
Welcome to Rome2Rio
--------------------
------Admin-----
3
Removing the Route Oporto-Lisbon
Routes:{Route{id := 1, duration := 5, destination := Location{id := 3, name := "Madrid", routes := {}}, transport := Transport{"Train"}, price := 30}, Route{id :=
number of routes:2
After removing:
Routes:{Route{id := 1, duration := 5, destination := Location{id := 3, name := "Madrid", routes := {}}, transport := Transport{"Train"}, price := 30}}
number of routes:1
```

7. Conclusions

With this project we have achieved the goals we set out to do. A platform where a regular user can search for all the possible ways of getting from point A to point B, but also where an admin of the platform can enter new routes and delete or edit existing routes.

For future work we could improve the project more on the emphasis of tests such as invariant tests which were not added along as improving the output of the all the possible routes to the User by sorting them by total cost or total time duration.

All the elements of the project worked with same effort to achieve its result and finished the task with approximately 20 hours of work.

8. References

1. Validated Designs for Object-oriented Systems, J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Springer, 2005
2. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
3. Overture tool web site, http://overturetool.org
4. MFES page at moodle, http://moodle.fe.up.pt