Modelo Formal de *Rome2Rio* em VDM++

Daniela José Antão João – up201505982

Diogo Henrique de Almeida Silva Pereira – up201505318

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

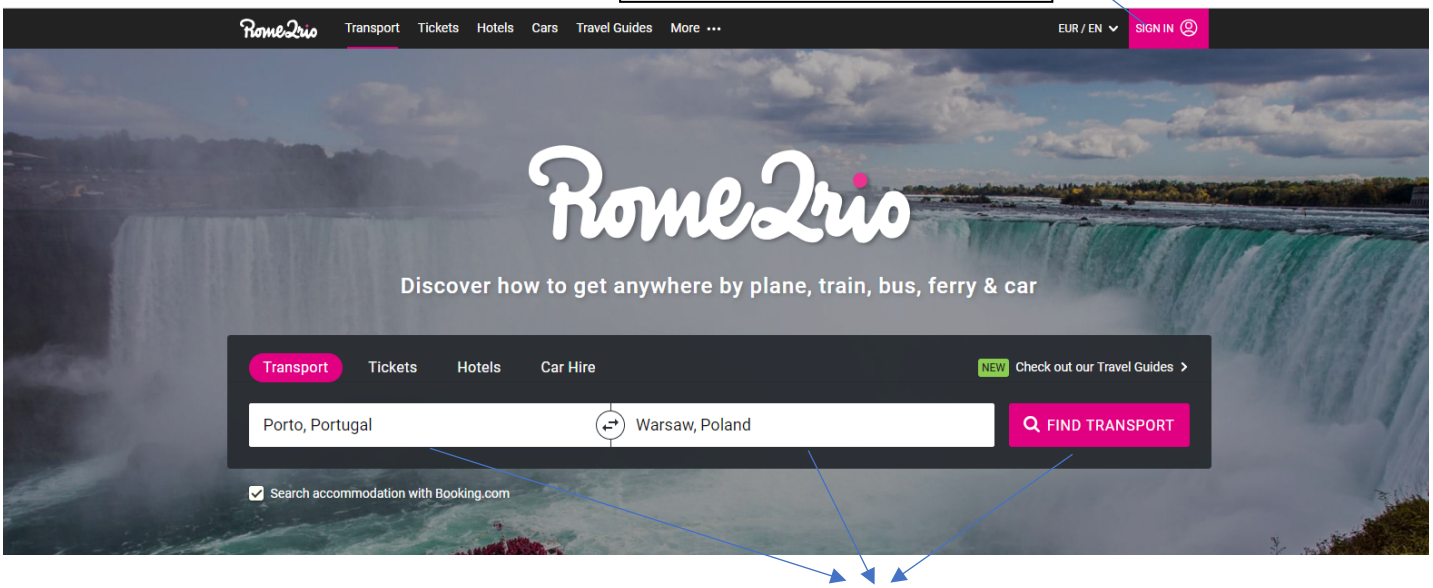
Ana Paiva

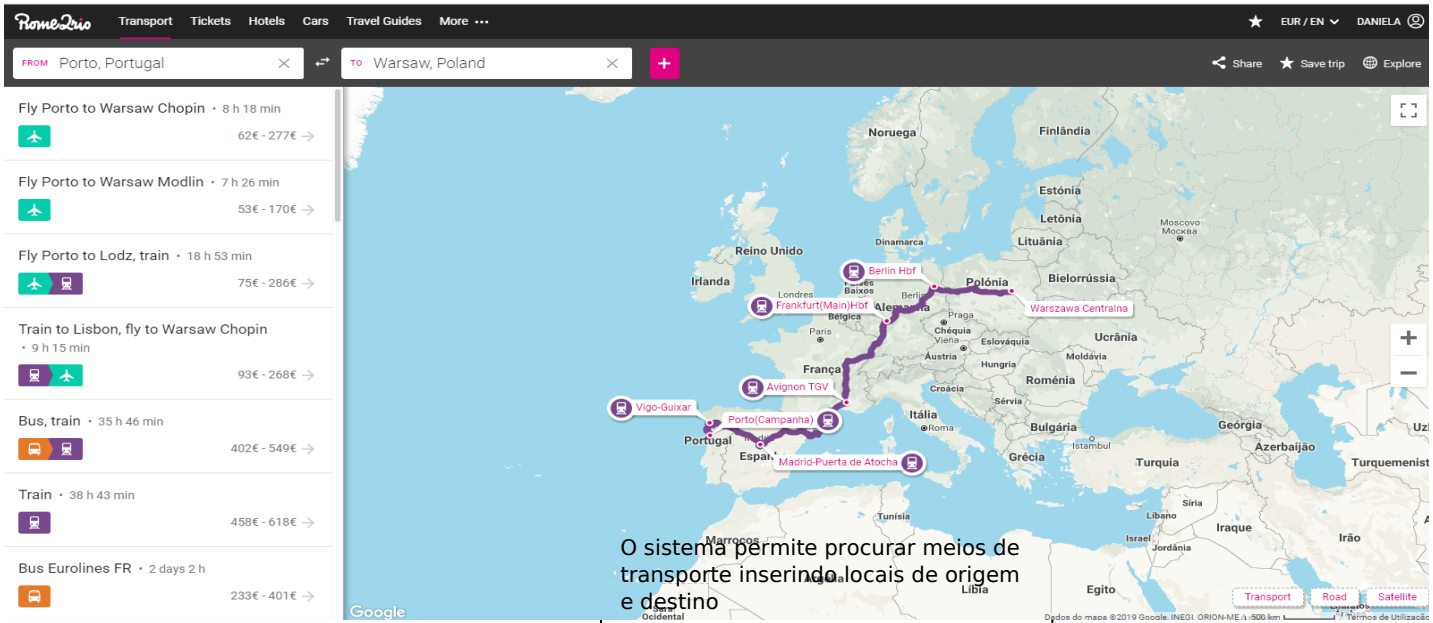
Indices

1. Descrição Informal do sistema e Lista de Requisitos

1.1 Descrição Informal do Sistema

O sistema permite ao utilizador registar-se e editar alguma informação pessoal





O sistema permite procurar meios de transporte inserindo locais de origem e destino

1.2 Lista de requisitos

ID - Nome	Prioridade	Descrição
R1 – Registo no Sistema	Obrigatório	Permitir que uma pessoa se possa registar na plataforma utilizando email, password e nome. Um utilizador não consegue criar uma conta se o email já estiver registado no sistema
R2- Iniciar sessão no sistema	Obrigatório	Permitir que um utilizador registado possa fazer login e logout
R3- Editar perfil	Obrigatório	Permitir que um utilizador altere o nome
R4- Visualizar perfil	Obrigatório	Permite que um utilizador possa visualizar o seu próprio perfil com o seu nome e com o nome que pretende que seja visível
R5- Alterar password	Obrigatório	Permite que o utilizador altere a sua password
R6 – Procurar Transporte de uma cidade para outra	Obrigatório	Permite que o utilizador encontre os transportes possíveis introduzindo a cidade de origem e a cidade de destino
R7 – Procurar Avião de uma cidade para outra	Obrigatório	Permite que o utilizador encontre os aviões disponíveis introduzindo a cidade de origem e a cidade de destino
R8-Procurar Comboio de uma cidade para outra	Obrigatório	Permite que o utilizador encontre os comboios disponíveis introduzindo a cidade de origem e a cidade de destino
R9 – Procurar autocarro de uma	Obrigatório	Permite que o utilizador encontre os

cidade para outra		comboios disponíveis introduzindo a cidade de origem e a cidade de destino
R10- Procurar todos os destinos possíveis a partir de uma cidade	Obrigatório	Permite que o utilizador encontre todos os destinos possíveis introduzindo a cidade de origem

2. Casos de Uso e Modelo UML

2.1 Casos de Uso

Cenário	**Registo de um utilizador**
Descrição	Um utilizador não registado pode registar-se para usufruir de todas as funcionalidades do sistema
Pré-condições	1. O email inserido para registo não pode encontrar-se entre os utilizadores registados 2. . O email inserido tem entre 1 a 255 caracteres 3. A password inserida tem entre 1 a 29 caracteres 4. O nome tem entre 1 a 49 caracteres
Pós-condições	1. O email inserido passa a encontrar-se nos utilizadores registados.
Passos	1. O utilizador insere o seu email. 2. O utilizador insere a sua password. 3. O utilizador insere o seu nome.

Cenário	**Iniciar sessão**
Descrição	Um utilizador registado pode iniciar sessão para usufruir das funcionalidades do sistema.
Pré-condições	1. O email inserido para o início de sessão encontra-se entre os utilizadores registados 2. Não existe nenhum utilizador com sessão iniciada

Pós-condições	1. Passa a existir a sessão iniciada do utilizador em questão
Passos	(não especificado)

Cenário	**Terminar Sessão**
Descrição	Um utilizador com sessão iniciada pode terminar sessão
Pré-condições	1. Existe um utilizador com sessão iniciada.
Pós-condições	1. O utilizador passa a ser indefinido.
Passos	(não especificado)

Cenário	**Visualizar perfil**
Descrição	Um utilizador pode visualizar o seu próprio perfil com o seu nome e com o nome que pretende que seja visível
Pré-condições	1. O utilizador deve ter sessão iniciada.
Pós-condições	(não especificado)
Passos	(não especificado)

Cenário	**Editar perfil**
Descrição	Um utilizador pode alterar o seu nome
Pré-condições	1. O utilizador deve ter sessão iniciada 2. O nome inserido tem entre 1 a 49 caracteres
Pós-condições	1. O nome passa a ser o novo nome inserido
Passos	1. Inserir novo nome 2. O programa passa a mostrar o novo nome inserido

Cenário	**Alterar password**
Descrição	Um utilizador pode alterar a sua password
Pré-condições	1. O utilizador deve ter sessão iniciada 2. O password inserido tem entre 1 a 29 caracteres.
Pós-condições	1. A password passa a ser

	a password inserida
Passos	(não especificado)

Cenário	Procurar Transporte de uma cidade para outra
Descrição	Permite que o utilizador encontre os transportes possíveis introduzindo a cidade de origem e a cidade de destino
Pré-condições	1. O utilizador deve ter sessão iniciada 2. A cidade de origem e destino devem existir no sistema
Pós-condições	(não especificado)
Passos	1. O utilizador introduz a cidade de origem e destino 2. O programa mostra todos os meios de transporte disponíveis

Cenário	**Procurar Avião de uma cidade para outra**
Descrição	Permite que o utilizador encontre os aviões disponíveis introduzindo a cidade de origem e a cidade de destino
Pré-condições	1. O utilizador deve ter sessão iniciada 2. A cidade de origem e destino devem existir no sistema
Pós-condições	(não especificado)
Passos	1. O utilizador introduz a cidade de origem e destino 2. O programa mostra todos os aviões disponíveis

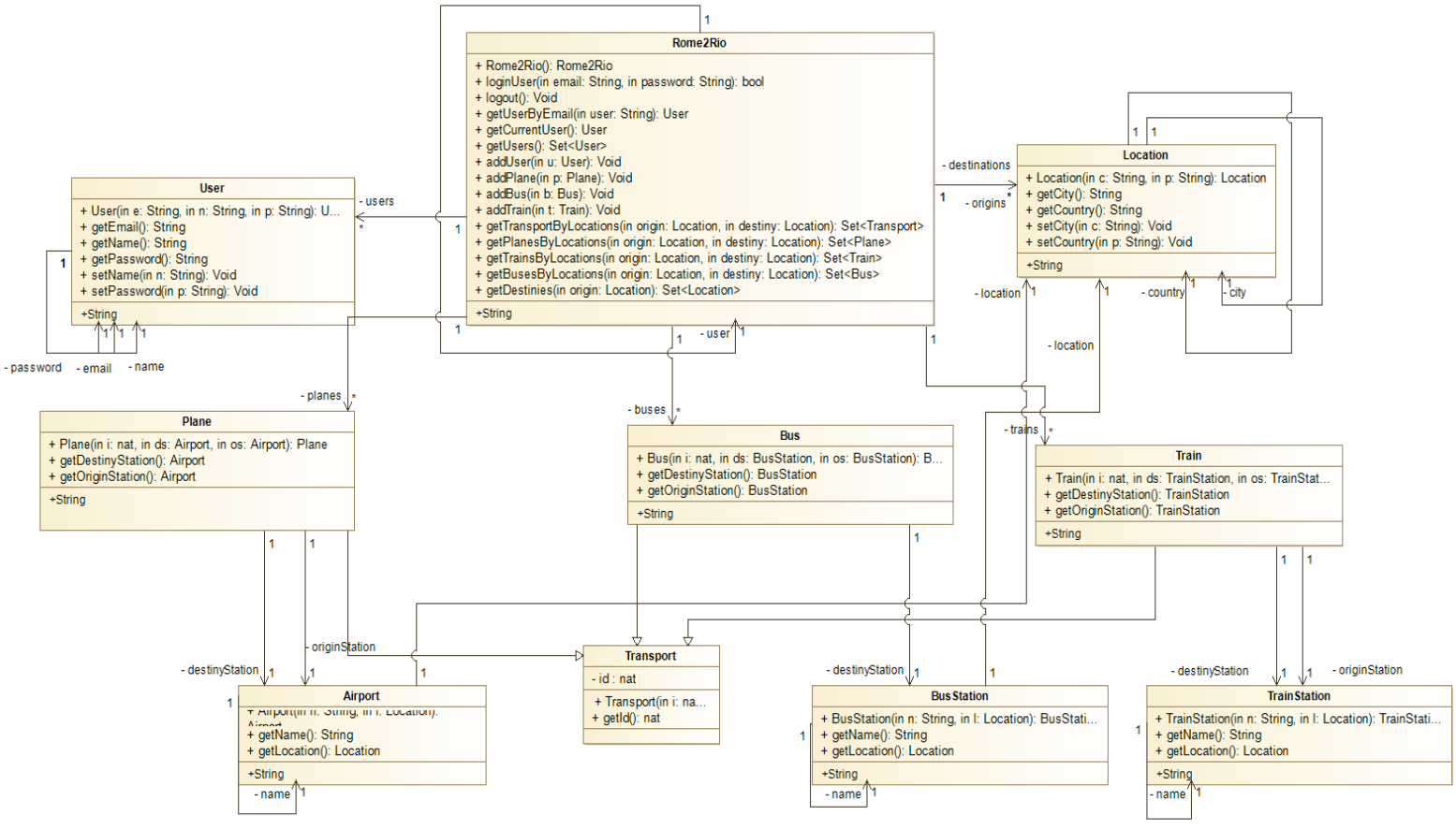
Cenário	**Procurar Comboio de uma cidade para outra**
Descrição	Permite que o utilizador encontre os comboios disponíveis introduzindo a cidade de origem e a cidade de destino
Pré-condições	1. O utilizador deve ter sessão iniciada 2. A cidade de origem e destino devem existir no sistema
Pós-condições	(não especificado)
Passos	1. O utilizador introduz a cidade

	de origem e destino 2. O programa mostra todos comboios disponíveis

Cenário	**Procurar autocarro de uma cidade para outra**
Descrição	Permite que o utilizador encontre os comboios disponíveis introduzindo a cidade de origem e a cidade de destino
Pré-condições	1. O utilizador deve ter sessão iniciada 2. A cidade de origem e destino devem existir no sistema
Pós-condições	(não especificado)
Passos	1. O utilizador introduz a cidade de origem e destino 2. O programa mostra todos os autocarros disponíveis

Cenário	**Procurar todos os destinos possíveis a partir de uma cidade**
Descrição	Permite que o utilizador encontre todos os destinos possíveis introduzindo a cidade de origem
Pré-condições	1. O utilizador deve ter sessão iniciada 2. A cidade de origem deve existir no sistema
Pós-condições	(não especificado)
Passos	1. O utilizador introduz a cidade de origem 2. O programa mostra todos os destinos possíveis a partir daquela cidade

2.1 Modelo UML



Classe	Descrição
Rome2Rio	Nesta classe são definidas as cidades de destino e origem, aviões, comboios e autocarros e utilizadores. Contém as funções de *login*, *logout* , pesquisa de transporte, etc.
Transport	Superclasse que define um transporte com o seu id.
Location	Classe que define uma localização como um par cidade-país.
User	Classe que define um utilizador com um email, nome, password.
Plane	Classe filha de *Transport* que define um avião tendo em conta o id, localização de origem e localização de destino.
Train	Classe filha de *Transport* que define um comboio tendo em conta o id, localização de origem e localização de destino.
Bus	Classe filha de *Transport* que define um autocarro tendo em conta o id, localização de origem e localização de destino.
TrainStation	Classe que define uma estação de comboios tendo em conta o nome e a sua localização
BusStation	Classe que define uma estação de autocarros tendo em conta o nome e a sua localização
Airport	Classe que define um aeroporto tendo em conta o nome e a sua localização

3. Modelo VDM++

3.1 Classe *User*

```
 class User

types
public String = seq of char;

values
-- TODO Define values here
instance variables
 email: String:="";
 name: String:="";
 password: String:="";

operations
public User : String*String*String ==> User
User (e, n, p)== (
      email:=e;
      name:=n;
      password:=p
      )

      pre len e > 0 and len e < 256
            and len p > 0 and len p < 30
```

```
            and len n > 0 and len n < 50;

            -- Get email of user
      pure public getEmail : () ==> String
      getEmail() == return email;

            -- Get name of user
      public getName : () ==> String
      getName() == return name;

      -- Get password of user
      public getPassword : () ==> String
      getPassword() == return password;

      -- Set name of user
      public setName: String ==> ()
      setName(n) == (
            name := n
      ;)
      post name = n;

      -- Set password of user
      public setPassword: String ==> ()
      setPassword(p) == (
            password:= p
      ;)
      post password = p;


end User
```

3.2 Classe *Transport*

```
class Transport
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
private id: nat;
operations

public Transport : nat ==> Transport
      Transport(i)==(
      id:=i;
      );

public getId: () ==> nat
            getId() == (return id);
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
      end Transport
```

3.3 Classe *TrainStation*

```
class TrainStation
types
public String = seq of char;
values
-- TODO Define values here
instance variables
private name:String;
private location: Location;

operations

-- Returns the name of the airport in the Airport object --

public TrainStation : String*Location ==> TrainStation
TrainStation (n, l)== (
      name:=n;
      location:=l;
      );

public getName: () ==> String
getName() == (return name);

public getLocation: () ==> Location
getLocation() == (return location);


      end TrainStation
```

3.4 Classe *Train*

```
class Train is subclass of Transport
types
public String = seq of char;
values
-- TODO Define values here
instance variables
private destinyStation: TrainStation;
private originStation: TrainStation;

operations
public Train : nat * TrainStation * TrainStation ==> Train
Train (i, ds, os)== (
      Transport(i);
      destinyStation:=ds;
      originStation:=os;
      );



public getDestinyStation: () ==> TrainStation
      getDestinyStation() == (return destinyStation);

public getOriginStation: () ==> TrainStation
```

```
      getOriginStation() == (return originStation);


      end Train
```

3.5 Classe *Plane*

```
class Plane is subclass of Transport
types
public String = seq of char;
values
-- TODO Define values here
instance variables
private destinyStation: Airport;
private originStation: Airport;

operations
public Plane : nat * Airport * Airport ==> Plane
Plane(i, ds, os)== (
      Transport(i);
      destinyStation:=ds;
      originStation:=os;
      );

public getDestinyStation: () ==> Airport
      getDestinyStation() == (return destinyStation);

public getOriginStation: () ==> Airport
      getOriginStation() == (return originStation);

end Plane
```

3.6 Classe *Bus*

```
class Bus is subclass of Transport
types
public String = seq of char;
values
-- TODO Define values here
instance variables
private destinyStation: BusStation;
private originStation: BusStation;

operations
public Bus : nat * BusStation * BusStation ==> Bus
Bus (i, ds, os)== (
      Transport(i);
      destinyStation:=ds;
      originStation:=os;
      );


public getDestinyStation: () ==> BusStation
      getDestinyStation() == (return destinyStation);

public getOriginStation: () ==> BusStation
      getOriginStation() == (return originStation);
```

```
end Bus
```

3.7 Classe *Airport*

```
class Airport
types
public String = seq of char;
values
-- TODO Define values here
instance variables
private name:String;
private location: Location;

operations

-- Returns the name of the airport in the Airport object --

public Airport : String*Location ==> Airport
Airport (n, l)== (
      name:=n;
      location:=l;
      );

public getName: () ==> String
getName() == (return name);

public getLocation: () ==> Location
getLocation() == (return location);

end Airport
```

3.8 Classe *BusStation*

```
class BusStation
types
public String = seq of char;
values
-- TODO Define values here
instance variables
private name: String;
private location: Location;

operations

-- Returns the name of the airport in the Airport object --

public BusStation : String*Location ==> BusStation
BusStation (n, l)== (
      name:=n;
      location:=l;
      );

public getName: () ==> String
```

```
getName() == (return name);

public getLocation: () ==> Location
      getLocation() == (return location);

end BusStation
```

3.9 Classe *Location*

```
class Location
      types
                  public String = seq of char;

      instance variables
                  private city : String;
                  private country : String;

                  inv city <> [];
                  inv country <> [];

      operations
                  -- Create a new Location object with all necessary
parameters --
                  public Location: String * String ==> Location
                        Location(c, p) == (city := c; country := p;
return self)
                  pre c <> [] and p <> [];

                  -- GETS --

                  -- Returns the name of the city in the Location
object --
                  public getCity: () ==> String
                        getCity() == (return city);

                  -- Return the name of the country in the Location
object --
                  public getCountry: () ==> String
                        getCountry() == (return country);

                  -- SETS --

                  -- Changes the name of the city in the Location
object --
                  public setCity: String ==> ()
                        setCity(c) == (city := c; return)
                        pre c <> [];

                  -- Changes the name of the country in the Location
object --
                  public setCountry: String ==> ()
                        setCountry(p) == (country := p; return)
                        pre p <> [];

end Location
```

3.10 Classe *Rome2Rio*

```
class Rome2Rio
types
public String = seq of char;

values
-- TODO Define values here

instance variables
users: set of User;
destinations: set of Location;
origins: set of Location;
planes: set of Plane;
buses: set of Bus;
trains: set of Train;
user: String;
operations

public Rome2Rio : () ==> Rome2Rio
Rome2Rio() == (
users:={};
destinations:={};
origins:={};
planes:={};
buses:={};
trains:={};
user := "undefined";
);

  /*******************************************************/
  /*****************     LOGIN & LOGOUT    ***************/
  /*******************************************************/

      -- Login in the application
      public loginUser : String * String ==> bool
      loginUser(email, password) ==
            if getUserByEmail(email).getPassword() = password
            then (
                  user := email;
                  return true;
            )
            else return false
      pre len email > 0 and len email < 50
            and user = "undefined";

      -- Logout from the application
      public logout : () ==> ()
      logout() ==
            user := "undefined"
      pre not user = "undefined";

      /*******************************************************/
      /*****************       Users Management *****************/
      /*******************************************************/

      --Get user by email
```

```
	public getUserByEmail : String ==> User
	getUserByEmail(user) == (
		for all u in set users do (
			if user = u.getEmail()
			then return u;
			);
		return new User();
	);

	--Get current logged in user
	public getCurrentUser : () ==> User
	getCurrentUser() == (
		return getUserByEmail(user);
	)
	pre user <> "undefined";

	-- Returns all registered users
	pure public getUsers : () ==> set of User
	getUsers() == return users;

	--Add a user
	public addUser: User ==> ()
	addUser(u) == (
		users := {u} union users
	);

	/*******************************************************/
	/*****************      Search  ******************/
	/*******************************************************/

	public addPlane: Plane ==> ()
	addPlane(p) == (
		planes:= {p} union planes;
		);

	public addBus: Bus ==> ()
	addBus(b) == (
		buses:= {b} union buses;
		);

	public addTrain: Train ==> ()
	addTrain(t) == (
		trains:= {t} union trains;
		);

	-- Search Transportation from a place to another--
	public getTransportByLocations: Location * Location ==> set of
Transport
	getTransportByLocations(origin, destiny)==(
	dcl availableTransportation : set of Transport :={};
	for all p in set planes do
		if (origin <> p.getOriginStation().getLocation() and
destiny <> p.getDestinyStation().getLocation())
		then availableTransportation:= availableTransportation
union {p};
	for all t in set trains do
		if (origin <> t.getOriginStation().getLocation() and
destiny <> t.getDestinyStation().getLocation())
		then availableTransportation:= availableTransportation
union {t};
	for all b in set buses do
```

```
            if (origin <> b.getOriginStation().getLocation() and
destiny <> b.getDestinyStation().getLocation())
            then availableTransportation:= availableTransportation
union {b};


      return availableTransportation
      )
  pre not user = "undefined"
  and {origin} subset origins
  and {destiny} subset destinations;

            -- Search Planes from a place to another--
      public getPlanesByLocations: Location * Location ==> set of
Plane
      getPlanesByLocations(origin, destiny)==(
      dcl availablePlanes : set of Plane :={};
      for all p in set planes do
            if (origin <> p.getOriginStation().getLocation() and
destiny <> p.getDestinyStation().getLocation())
            then availablePlanes:= availablePlanes union {p};

      return availablePlanes
      )
  pre not user = "undefined"
  and {origin} subset origins
  and {destiny} subset destinations;

                  -- Search Trains from a place to another--
      public getTrainsByLocations: Location * Location ==> set of
Train
      getTrainsByLocations(origin, destiny)==(
      dcl availableTrains : set of Train :={};
      for all t in set trains do
            if (origin <> t.getOriginStation().getLocation() and
destiny <> t.getDestinyStation().getLocation())
            then availableTrains:= availableTrains union {t};

      return availableTrains
      )
  pre not user = "undefined"
  and {origin} subset origins
  and {destiny} subset destinations;

                        -- Search Buses from a place to another--
      public getBusesByLocations: Location * Location ==> set of Bus
      getBusesByLocations(origin, destiny)==(
      dcl availableBuses : set of Bus :={};
      for all b in set buses do
            if (origin <> b.getOriginStation().getLocation() and
destiny <> b.getDestinyStation().getLocation())
            then availableBuses:= availableBuses union {b};

      return availableBuses
      )
  pre not user = "undefined"
  and {origin} subset origins
  and {destiny} subset destinations;
```

```
      -- Search all destinies from a place r--
            public getDestinies: Location ==> set of Location
      getDestinies(origin)==(
      dcl availableDestinies : set of Location :={};
      for all p in set planes do
            if origin <> p.getOriginStation().getLocation()
            then availableDestinies:= availableDestinies union
{p.getDestinyStation().getLocation()};
      for all t in set trains do
            if origin <> t.getOriginStation().getLocation()
            then availableDestinies:= availableDestinies union
{t.getDestinyStation().getLocation()};
      for all b in set buses do
            if origin <> b.getOriginStation().getLocation()
            then availableDestinies:= availableDestinies union
{b.getDestinyStation().getLocation()};


      return availableDestinies
      )
  pre not user = "undefined"
  and {origin} subset origins;


end Rome2Rio
```

4. Validação do Modelo

4.1 Classe *TestSuiteRome2Rio*

```
class TestSuiteRome2Rio
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:\n");
			IO`print("expected (");
			IO`print(expected);
			IO`println(")\n")
			)
			post expected = actual


end TestSuiteRome2Rio


class TestRome2Rio
	operations
		public static main: () ==> ()
		main() ==
		(
			new AirportTest().main();
			new BusTest().main();
			new BusStationTest().main();
			new DateTimeTest().main();
			new LocationTest().main();
			new PlaneTest().main();
			new TrainTest().main();
			new TrainStationTest().main();
			new TransportTest().main();
			new UserTest().main();
			new Rome2RioTest().main();
		);

end TestRome2Rio
```

4.2 Testes da Classe *Airport*

```
class AirportTest is subclass of TestSuiteRome2Rio

	instance variables
	l1 :  Location := new Location("Porto", "Portugal");
	l2 : Location := new Location("Madrid", "Spain");
	a1 : Airport := new Airport("Aeroporto do Porto", l1);
	a2 : Airport := new Airport("Aeroporto de Madrid", l2);

	operations
		private testgetName: () ==> ()
			testgetName() == (
				assertEqual(a1.getName(), "Aeroporto de
Lisboa");
				assertEqual(a2.getName(), "Aeroporto do
Porto");
			);
```

```
                  private testgetLocation: () ==> ()
                  testgetLocation() == (
                        assertEqual(a1.getLocation(), "Lisboa");
                        assertEqual(a2.getLocation(), "Porto");
                  );


      public static main: () ==> ()
                  main() ==
                  (
                    dcl test : AirportTest := new AirportTest();
                    test.testgetName();
                    test.testgetLocation();
                    );

end AirportTest
```

4.3 Testes da Classe *Bus*

```
class BusTest is subclass of TestSuiteRome2Rio

      instance variables
      l1 :  Location := new Location("Porto", "Portugal");
      l2 : Location := new Location("Madrid", "Spain");
      bs1 : BusStation := new BusStation("Aliados", l1);
      bs2 : BusStation := new BusStation("IPO", l1);
      bs3 : BusStation := new BusStation("Santa Catarina", l1);
      bs4 : BusStation := new BusStation("Trindade", l1);
      b1 : Bus := new Bus(105, bs1, bs2);
      b2 : Bus := new Bus(205, bs3, bs4);

      operations
            private testgetDestinyStation: () ==> ()
                  testgetDestinyStation() == (
                        assertEqual(b1.getDestinyStation(), bs1);
                        assertEqual(b2.getDestinyStation(), bs3);
                  );

                  private testgetOriginStation: () ==> ()
                  testgetOriginStation() == (
                        assertEqual(b1.getOriginStation(), bs2);
                        assertEqual(b2.getOriginStation(), bs4);
                  );


      public static main: () ==> ()
                  main() ==
                  (
                    dcl test : BusTest := new BusTest();
                    test.testgetDestinyStation();
                    test.testgetOriginStation();
                    );

end BusTest
```

4.4 Testes da Classe *BusStation*

```
class BusStationTest is subclass of TestSuiteRome2Rio

	instance variables
	l1 :  Location := new Location("Porto", "Portugal");
	l2 : Location := new Location("Madrid", "Spain");
	bs1 : BusStation := new BusStation("Aliados", l1);
	bs2 : BusStation := new BusStation("IPO", l1);
	bs3 : BusStation := new BusStation("Santa Catarina", l1);
	bs4 : BusStation := new BusStation("Trindade", l1);

	operations
		private testgetName: () ==> ()
			testgetName() == (
				assertEqual(bs1.getName(), "Aliados");
				assertEqual(bs2.getName(), "IPO");
				assertEqual(bs3.getName(), "Santa Catarina");
				assertEqual(bs4.getName(), "Trindade");
			);

			private testgetLocation: () ==> ()
			testgetLocation() == (
				assertEqual(bs1.getLocation(), l1);
				assertEqual(bs2.getLocation(), l1);
				assertEqual(bs3.getLocation(), l1);
				assertEqual(bs4.getLocation(), l1);
			);


	public static main: () ==> ()
		    main() ==
		    (
		      dcl test : BusStationTest := new BusStationTest();
		      test.testgetName();
		      test.testgetLocation();
		      );

end BusStationTest
```

4.5 Testes da Classe *Location*

```
class LocationTest is subclass of TestSuiteRome2Rio

	instance variables
	l1 :  Location := new Location("Porto", "Portugal");
	l2 : Location := new Location("Madrid", "Spain");

	operations
		private testgetCity: () ==> ()
			testgetCity() == (
				assertEqual(l1.getCity(), "Porto");
				assertEqual(l2.getCity(), "Madrid");
			);

			private testgetCountry: () ==> ()
```

```
                testgetCountry() == (
                      assertEqual(l1.getCountry(), "Portugal");
                      assertEqual(l2.getCountry(), "Spain");
                );

                private testsetCity: () ==> ()
                testsetCity() == (
                      l1.setCity("Lisbon");
                      l2.setCity("Barcelona");
                );

                private testsetCountry: () ==> ()
                testsetCountry() == (
                      l1.setCountry("Lusitanos");
                      l2.setCountry("Mouros");
                );


      public static main: () ==> ()
                main() ==
                (
                  dcl test : LocationTest := new LocationTest();
                  test.testgetCity();
                  test.testgetCountry();
                  test.testsetCity();
                  test.testsetCountry();

                  );

end LocationTest
```

4.6 Testes da Classe *Plane*

```
class PlaneTest is subclass of TestSuiteRome2Rio

      instance variables
      l1 :  Location := new Location("Porto", "Portugal");
      l2 : Location := new Location("Madrid", "Spain");
      a1 : Airport := new Airport("Aeroporto do Porto", l1);
      a2 : Airport := new Airport("Aeroporto de Madrid", l2);
      p1 : Plane := new Plane(101, a1, a2);
      p2 : Plane := new Plane(100, a2, a1);

      operations
            private testgetDestinyStation: () ==> ()
                  testgetDestinyStation() == (
                        assertEqual(p1.getDestinyStation(), a1);
```

```
                        assertEqual(p2.getDestinyStation(), a2);
                    );

                    private testgetOriginStation: () ==> ()
                    testgetOriginStation() == (
                        assertEqual(p1.getOriginStation(), a2);
                        assertEqual(p2.getOriginStation(), a1);
                    );


        public static main: () ==> ()
                    main() ==
                    (
                      dcl test : PlaneTest := new PlaneTest();
                      test.testgetDestinyStation();
                      test.testgetOriginStation();
                      );

end PlaneTest
```

4.7 Testes da Classe *Rome2Rio*

4.8 Testes da Classe *Train*

```
class TrainTest is subclass of TestSuiteRome2Rio

        instance variables
        l1 :  Location := new Location("Porto", "Portugal");
        l2 : Location := new Location("Madrid", "Spain");
        ts1 : TrainStation := new TrainStation("Campanha", l1);
        ts2 : TrainStation := new TrainStation("Espinho", l2);
        ts3 : TrainStation := new TrainStation("Trindade", l1);
        ts4 : TrainStation := new TrainStation("Estarreja", l2);
        t1 : Train := new Train(76, ts1, ts2);
        t2 : Train := new Train(66, ts3, ts4);


        operations
              private testgetDestinyStation: () ==> ()
                    testgetDestinyStation() == (
                        assertEqual(t1.getDestinyStation(), ts1);
                        assertEqual(t2.getDestinyStation(), ts2);
                    );

                    private testgetOriginStation: () ==> ()
                    testgetOriginStation() == (
                        assertEqual(t1.getOriginStation(), ts3);
                        assertEqual(t2.getOriginStation(), ts4);
                    );
```

```
	public static main: () ==> ()
		main() ==
		(
		  dcl test : TrainTest := new TrainTest();
		  test.testgetDestinyStation();
		  test.testgetOriginStation();
		  );

end TrainTest
```

4.9 Testes da Classe *TrainStation*

```
class TrainStationTest is subclass of TestSuiteRome2Rio

	instance variables
	l1 :  Location := new Location("Porto", "Portugal");
	l2 : Location := new Location("Madrid", "Spain");
	l3 : Location := new Location("Aveiro", "Portugal");
	ts1 : TrainStation := new TrainStation("Campanha", l1);
	ts2 : TrainStation := new TrainStation("Espinho", l2);
	ts3 : TrainStation := new TrainStation("Trindade", l1);
	ts4 : TrainStation := new TrainStation("Estarreja", l2);


	operations
		private testgetName: () ==> ()
			testgetName() == (
				assertEqual(ts1.getName(), "Campanha");
				assertEqual(ts2.getName(), "Espinho");
				assertEqual(ts3.getName(), "Trindade");
				assertEqual(ts4.getName(), "Estarreja");
			);

			private testgetLocation: () ==> ()
			testgetLocation() == (
				assertEqual(ts1.getLocation(), l1);
				assertEqual(ts2.getLocation(), l2);
				assertEqual(ts3.getLocation(), l1);
				assertEqual(ts4.getLocation(), l2);
			);


	public static main: () ==> ()
		main() ==
		(
		  dcl test : TrainStationTest := new
TrainStationTest();
		  test.testgetName();
		  test.testgetLocation();
		  );

end TrainStationTest
```

4.10 Testes da Classe *Transport*

```
class TransportTest is subclass of TestSuiteRome2Rio

      instance variables
      tr1 : Transport := new Transport(105);
      tr2 : Transport := new Transport(101);


      operations
            private testgetId: () ==> ()
                  testgetId() == (
                        assertEqual(tr1.getId(), 105);
                        assertEqual(tr2.getId(), 101);
                  );


      public static main: () ==> ()
                main() ==
                (
                  dcl test : TransportTest := new TransportTest();
                  test.testgetId();
                  );

end TransportTest
```

4.11 Testes da Classe *User*

```
class UserTest is subclass of TestSuiteRome2Rio

      instance variables
      u1 :  User := new User("danjoao@gmail.com", "Malhoa", "123");
      u2 : User := new User("diogohp@gmail.com", "Palhas", "simples");

      operations
            private testgetEmail: () ==> ()
                  testgetEmail() == (
                        assertEqual(u1.getEmail(),
"danjoao@gmail.com");
                        assertEqual(u2.getEmail(),
"diogohp@gmail.com");
                  );

                  private testgetName: () ==> ()
                  testgetName() == (
                        assertEqual(u1.getName(), "Malhoa");
                        assertEqual(u2.getName(), "Palhas");
                  );

                  private testgetPassword: () ==> ()
                  testgetPassword() == (
                        assertEqual(u1.getPassword(), "123");
                        assertEqual(u2.getPassword(), "simples");
                  );
```

```
            private testsetName: () ==> ()
                testsetName() == (
                      u1.setName("Ana Malhoa");
                      u2.setName("Palhinhas");
                );

                private testsetPassword: () ==> ()
                testsetPassword() == (
                      u1.setPassword("321");
                      u2.setPassword("dificil");
                );


      public static main: () ==> ()
                main() ==
                (
                  dcl test : UserTest := new UserTest();
                  test.testgetEmail();
                  test.testgetName();
                  test.testsetName();
                  test.testsetPassword();
                  );

end UserTest
```

5. Model verification

5.1 Example of postcondition verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
8	User`setPassword(String)	Operation establishes postcondition

```
-- Set password of user

public setPassword: String ==> ()

setPassword(p) == (

      password:= p

;)

post password = p;
```

This simple verification is done on every set operation. It verifies that the value that was updated is in fact the value that was supplied to the operation.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

```
 No.   PO Name                              Type
 1     DateTime`DateTime(nat,nat,nat,na     state invariant holds
       t,nat)
```

The relevant invariant under analysis is:

```
  -- Creates a new Date object given its year, month and day --
public DateTime: nat * nat * nat * nat * nat ==> DateTime

DateTime(y, m, d, h, min) == (year := y; month := m; day := d; hour :=
h; minutes := min; return self)

pre y > 0 and m > 0 and m <= 12 and d > 0 and d <= 31 and h < 24 and m
< 60;
```

Due to the preconditions present in the constructor of this class. The invariant that enforces the date to be valid will always hold. Any other attempt to create a non-valid date will result in the program not running.

6. Conclusão

O modelo efetuado cobre a maior parte das especificações pedidas.

Com mais tempo, teríamos colocado mais pré-condições e pós-condições. Teríamos também melhorado o modelo de forma a não ter um diagrama de classes tão amplo.

O projeto demorou cerca de 30 horas a ser desenvolvido.

O trabalho foi elaborado de forma igualmente distribuída pelos dois elementos do grupo.

7. Referencias

1. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
2. Overture tool web site, http://overturetool.org
3. https://www.modelio.org/