Green Way System

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Ana Rita Ferreira - 201205014/ei12052@fe.up.pt
Jorge Teixeira - 201205117/ei12030@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

13 de Dezembro de 2015

Conteúdo

1 Informal system description and list of requirements

Requirements should include any relevant constraints (regarding safety, etc.). Each requirement should have an identifier. You may have optional requirements.

2 Visual UML model

A use case model, describing the system actors and use cases, with a short description of each major use case. One or more class diagram(s), describing the structure of the VDM++ model, with a short description of each class, plus any other relevant explanations.

3 Formal VDM++ model

VDM++ classes, properly commented. Needed data types (e.g., String, Date, etc.) should be modeled with types, values and functions. Domain entities should be modeled with classes, ins tance variables and operations. You are expected to make adequate usage of the VDM++ types (sets, sequences, maps, etc.) and create a model at a high level of abstraction. The model should contain adequate contracts, i.e., invariants, preconditions, and -conditions. Post-conditions need only be defined in cases where they are significantly different from the operation or function body (e.g., t he post-condition of a sqrt(x) operation, which simply states that x = RESULT * RESULT, should be significantly different than the body); for learning purposes, you should define post-conditions for at least two operations. During the development of the project, if you foresee that the size of the VDM++ model will be less than 5 pages (or 7.5 pages in case of groups of 3 students) or more than 10 pages (or 15 pages in case of groups of 3 students), you should contact your teacher to possibly adjust the scope of the system or the modeling approach being followed.

3.1 Client

```
class Client

instance variables
 public name : seq1 of char;
 public payment_card : int;

operations


 public Client : seq1 of char * int ==> Client
  Client(nome, pay_card) == (
  name := nome;
  payment_card := pay_card;
  return self;
 );


 pure public getName: () ==> seq1 of char
 getName() == return name;


 pure public getCardNumber: () ==> int
 getCardNumber() == return payment_card;

end Client
```

3.2 Green_Way

```
class Green_Way
```

```
types
   public Name = seq1 of char;

   public OriginDestiny :: origin: Highway
               destination: Highway;

  public Invoice :: month: nat1
      year: nat1
      price: real
      paid: bool;


instance variables

  public highway_prices : map OriginDestiny to real := { |-> };
 private sproviders: set of Service_Provider := { };
 public clients : set of Client := {};
 public passages: map Client to seq of Passage := { |-> };
 public invoices: map Client to seq of Invoice := { |-> };
 public month: nat1 := 1;

 public year: nat1 := 2015;

 inv month >= 1 and month <= 12;

operations

 --adiciona um cliente

  public addClient(client: Client) ==
  (
    clients := clients union {client};
    passages := passages munion {client |-> []};
    invoices := invoices munion {client |-> []};
  )
  pre
   not client in set clients
  post
    clients <> {};

  --remove um cliente
  public removeClient(client: Client) ==
  (
  clients := clients \ {client};
  )
  pre
   client in set clients;

  --adiciona um protocolo com um fornecedor de servicos
 public addServiceProvider(provider: Service_Provider) == (

  --MyTestCase`assertTrue(
    -- provider.getAllSpots() inter getAllSpots() = {}
    --);

  sproviders := sproviders union {provider};

 )
 pre
  not provider in set sproviders;

 --remove uma parceria com um fornecedor de servicos

 public removeServiceProvider(provider: Service_Provider) ==
  (
  sproviders := sproviders \ {provider};
  )
  pre
   provider in set sproviders;
  public getAllSpots : () ==> set of Spot
 getAllSpots() == (
   dcl all_spots: set of Spot := {};

   for all sprovider in set sproviders do
     all_spots := all_spots union sprovider.getAllSpots();

   return all_spots;
 );


 public getServiceProviderBySpot : Spot ==> Service_Provider
  getServiceProviderBySpot(spot) == (

  dcl all_spots : set of Spot := {};
  dcl sprovider: Service_Provider;
  dcl found: bool;
  found:= false;

  for all sp in set sproviders do(
   all_spots := sp.getAllSpots();
   if(spot in set all_spots) then(
     found:= true;
    sprovider:= sp;
    )
  );
   return sprovider;
  );


  public getLastPassage : Client * Service_Provider ==> [Passage]
  getLastPassage(client, sp) == (
  dcl client_passages : seq of Passage;
  dcl last_passage: [Passage] := nil;

   client_passages := passages(client);

   for all c in set elems client_passages do
    if (sp = c.provider) then
     last_passage := c;

   return last_passage;
 );


 public getLastHighwayPassage : Client ==> [Passage]
 getLastHighwayPassage(client) == (

   for r in reverse passages(client) do
    if (isofclass(Highway, r.spot)) then
     return r;
   return nil;
 );


 public incrementMonth : () ==> ()
  incrementMonth() == (

   if(month = 12) then(
    month:= 1;
    year := year + 1;)
   else
    month := month + 1;

   for all client in set clients do
    sendInvoice(client);
  );
```

```
public sendInvoice : Client ==> ()
sendInvoice(client) == (
  dcl client_balance: real := 0.0;
  dcl previous_month: nat1 := month;
  dcl previous_year : nat1 := year;

  if(month = 1) then(
   previous_month := 12;
   previous_year := year - 1;
  )
  else(
   previous_month := month - 1
  );

  for all p in set elems passages(client) do
   if(p.time.month = previous_month and p.time.year = previous_year) then
    client_balance := client_balance + p.cost;

  invoices := invoices ++ {client |-> invoices(client) ^ [mk_Invoice(previous_month,
      previous_year, client_balance, false)]};
);


public payInvoice : Client * nat1 * nat1==> ()
payInvoice(client, m, y) == (
 dcl client_invoices : seq of Invoice := invoices(client);

 for index = len client_invoices to 1 do
 if(client_invoices(index).month = m and client_invoices(index).year = y) then
 (
  client_invoices := client_invoices ++ { index |-> mk_Invoice(client_invoices(index).month,
       client_invoices(index).year, client_invoices(index).price, true)};
  invoices := invoices ++ {client |-> client_invoices};
  return;
 );
)
pre
 exists i in set elems invoices(client) & (i.month = m and i.year = y);


public passa(client: Client, s : Spot, time: Time) == (
 dcl estab : [Service_Provider] := nil;
 dcl cost : real;
 dcl new_passage : Passage;
 dcl last_passage : [Passage] := nil;
 dcl client_passages : seq of Passage;
 time.month := month;
 time.year := year;

 MyTestCase`assertTrue(
  (
   (
    isofclass(Highway, s)
   and
   (
    let n  = narrow_(s,Highway) in
    (
     (
      n.type = <EXIT>
      and
      getLastHighwayPassage(client) <> nil
      and
      let n1 = narrow_(getLastHighwayPassage(client).spot, Highway) in
      n1.type = <ENTRANCE>
      and
      mk_OriginDestiny(n1, n) in set dom highway_prices
     )
     or
     (
      n.type = <ENTRANCE>
```

```
       )
      )
     )
    )
    or (not isofclass(Highway, s))

   )
  );

  if(len passages(client) > 0) then
   last_passage := passages(client)(len passages(client));

  MyTestCase`assertTrue(
   last_passage = nil
   or
   (
    last_passage <> nil
    and
    time.timer - last_passage.time.timer > 0
    and
     (distance(s.local, last_passage.spot.local) / (time.timer - last_passage.time.timer)) <=
         MAX_SPEED
     and time.timer > last_passage.time.timer
    )
  );

   if (isofclass(Highway, s)) then
  (
   dcl hn : Highway;
   last_passage := getLastHighwayPassage(client);
   hn := s;
   if(hn.type = <ENTRANCE>) then
    (
    cost:= 0.0;
    )
   else
   (
    dcl entrance_node : Highway;
    entrance_node := last_passage.spot;
    cost := highway_prices(mk_OriginDestiny(entrance_node , hn));
   );
  )
   else
   (
    estab := getServiceProviderBySpot(s);

    last_passage := getLastPassage(client, estab);

    cost := estab.passa(client, s, time, last_passage)
   );

  client_passages := passages(client);
  new_passage := new Passage(client, s, time, estab, cost);
  client_passages := client_passages ^ [new_passage];
  passages := passages ++ { client |-> client_passages };
 )
 pre
 (
  client in set clients
 );

functions

 public distance(l1: Spot`Local, l2: Spot`Local) res: real ==
    (
      MATH`sqrt((l2.latitude - l1.latitude)*(l2.latitude - l1.latitude) + (l2.longitude - l1.
          longitude)*(l2.longitude - l1.longitude))
    );

end Green_Way
```

3.3 Highway

```
class Highway is subclass of Spot

types
 public Type = <ENTRANCE> | <EXIT>;

instance variables
 public type: Type;

operations

 public Highway : nat1 * nat1 * Type ==> Highway
  Highway(lat, long, t) == (
   local := mk_Local(lat, long);
   type := t;
  return self;
 );

end Highway
```

3.4 OnePassage

```
class OnePassage is subclass of Service

instance variables
 public spot: Spot;
 public price: real;

operations

 public OnePassage : Spot * real ==> OnePassage
  OnePassage(s, p) == (
   spot := s;
   price := p;
  return self;
 );


 public getAllSpots : () ==> set of Spot
 getAllSpots() == (
    return {spot};
 );


 public passa : Client * Spot * Time * [Passage] ==> real
 passa(-, -, -, -) == (
  return price;
 );

end OnePassage
```

3.5 Passage

```
class Passage

instance variables
 public client: Client;
 public spot: Spot;
 public time: Time;
 public provider: [Service_Provider];
 public cost: real;

operations

 public Passage : Client * Spot * Time * [Service_Provider] * real ==> Passage
  Passage(cl, spt, t, sprovider, cst) == (
   client := cl ;
  spot := spt;
  time := t;
  provider := sprovider;
  cost := cst;
  return self;
 )
 pre
 (
  (
   sprovider = nil
   and
   isofclass(Highway, spt)
  )
  or
   sprovider <> nil
 );
end Passage
```

3.6 Service

```
class Service

operations

 public getAllSpots : () ==> set of Spot
 getAllSpots() ==
    is subclass responsibility;


 public passa : Client * Spot * Time * [Passage] ==> real
 passa(client, spot, time, last_passage) ==
  is subclass responsibility;

end Service
```

3.7 Service_Provider

```
class Service_Provider

instance variables
 private name: seq1 of char;
 private services: set of Service := {};

operations

 public Service_Provider : seq1 of char ==> Service_Provider
 Service_Provider(n) == (
  name := n;
  return self
 );


 public addService : Service ==> ()
 addService(serv) == (

    services := services union {serv};
 );


 public getAllSpots : () ==> set of Spot
 getAllSpots() == (
    dcl spots: set of Spot := {};

    for all service in set services do
     spots := spots union service.getAllSpots();

   return spots;
 );

 --retorna o custo da passagem

 public passa : Client * Spot * Time * [Passage] ==> real
 passa(client, spot, time, last_passage) == (

   dcl all_spots : set of Spot := {};
   dcl used_service : Service;

    for all service in set services do(
    all_spots := service.getAllSpots();
    if(spot in set all_spots) then
     used_service := service;
   );

   return used_service.passa(client, spot, time, last_passage);
 );

end Service_Provider
```

3.8 Spot

```
class Spot
types
 public Local ::  latitude: nat1
          longitude: nat1;

instance variables
 public local: Local;

operations

 public Spot : nat1 * nat1 ==> Spot
  Spot(lat, long) == (
   local := mk_Local(lat, long);
  return self;
 );

end Spot
```

3.9 Time

```
class Time
instance variables

public timer: nat1;
public month: nat1;
public year: nat1;

operations


public static Diff : Time * Time ==> real
Diff (time1,time2) == (
  return time1.timer - time2.timer;
);


public Time : nat1 ==> Time
  Time(t) == (
   month := 1;
   year := 2015;
   timer := t;
  return self;
 );

functions


 public compareTimes : nat1 * nat1 -> int
  compareTimes(time1, time2) == (
  time1- time2
 );

end Time
```

4 Model validation (i.e., testing)

4.1 MyTestCase

```
class MyTestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

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
```

VDM++ test classes, containing adequate and thoroug h test cases defined by means of operations or traces. o Evidences of test results (passed/failed) and test coverage. It is sufficient to present the system classes mentioned in 4 painted with coverage information. Ideally, 100erage should be achieved. Optionally, figures of examples exercised in the test cases. Requirements traceability relationship between test cases and requirements. Ideally, 100% requirements coverage should be achieved. It is sufficient to indicate in com- ments the requirements that are exercised by each test.

5 Model verification (i.e., consistency analysis)

An example of domain verification, i.e., a proof sketch that a pre-condition of an op- erator, function or operation is not violated. You should present the proof obligation generated by the tool and your proof sketch. An example of invariant verification, i.e., a proof sketch that the body of an operation preserves invariants. You should present the proof obligation generated by the tool and your proof sketch.

6 Code generation

You should try to generate Java code from the VDM++ model and try to execute or test the generated code. Here you should describe the steps followed and results achieved.

7 Conclusions

Results achieved Things that could be improved Division of effort and contributions between team members

8 References