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

8 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.

4 Model validation (i.e., testing)

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

9 Charge

```
class Charge
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Charge
```

10 Client

```
class Client
types
 public ClientName = seq1 of char;
 public ClientCardNumber = seq1 of nat;

values
-- TODO Define values here

instance variables
 private name: ClientName;
 private cardNumber : ClientCardNumber;

operations

 public Client : ClientName * ClientCardNumber ==> Client
 Client(n, cn) == (
  name := n;
  cardNumber := cn;
  return self
 );


 pure public getName: () ==> ClientName
 getName() == return name;


 pure public getCardNumber: () ==> seq1 of nat
 getCardNumber() == return cardNumber;

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Client
```

11 Device

```
class Device
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Device
```

12 Green_Way

```
class Green_Way
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Green_Way
```

13 Passage

```
class Passage
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Passage
```

14 Protocol

```
class Protocol
types
-- TODO Define types here
```

```
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Protocol
```

15 Service

```
class Service
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Service
```

16 Service_Provider

```
class Service_Provider
types
 public SPName = seq1 of char;

values
-- TODO Define values here
instance variables
 private name: SPName;
 private services: set of Service := {};

operations

 public Service_Provider : SPName ==> Service_Provider
 Service_Provider(n) == (
  name := n;
  return self
 );


 pure public getName: () ==> SPName
 getName() == return name;



functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Service_Provider
```