Electronic Voting System

Assignment 5 (T05)

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação
Métodos Formais em Engenharia de *Software*
EIC0039-1S

João de Sá Balão Calisto Correia - 201208114 (ei12159@fe.up.pt)
João Pedro Matos Teixeira Dias - 201106781 (ei11137@fe.up.pt)
Eduardo José Valadar Martins - 201104191 (ei11104@fe.up.pt)

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

December 13, 2014

Contents

1 Informal system description and list of requirements

1.1 Informal system description

1.2 List of requirements

2 Visual UML Model

2.1 Use case model

2.2 State machine model

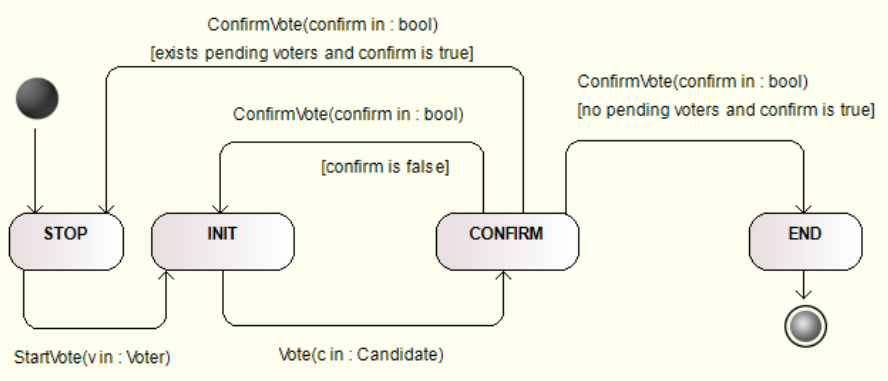

Figure 1: State Machine Diagram (Vote Process).

2.3 Class model

3 Formal VDM++ Model

3.1 Candidate

```
class Candidate
instance variables
 public name: seq of char;
operations
--Constructor-Create a new Candidate

 public Candidate: seq of char ==> Candidate
 Candidate(n) == (name := n; return self)
 pre name <> []
 post name = n
end Candidate
```

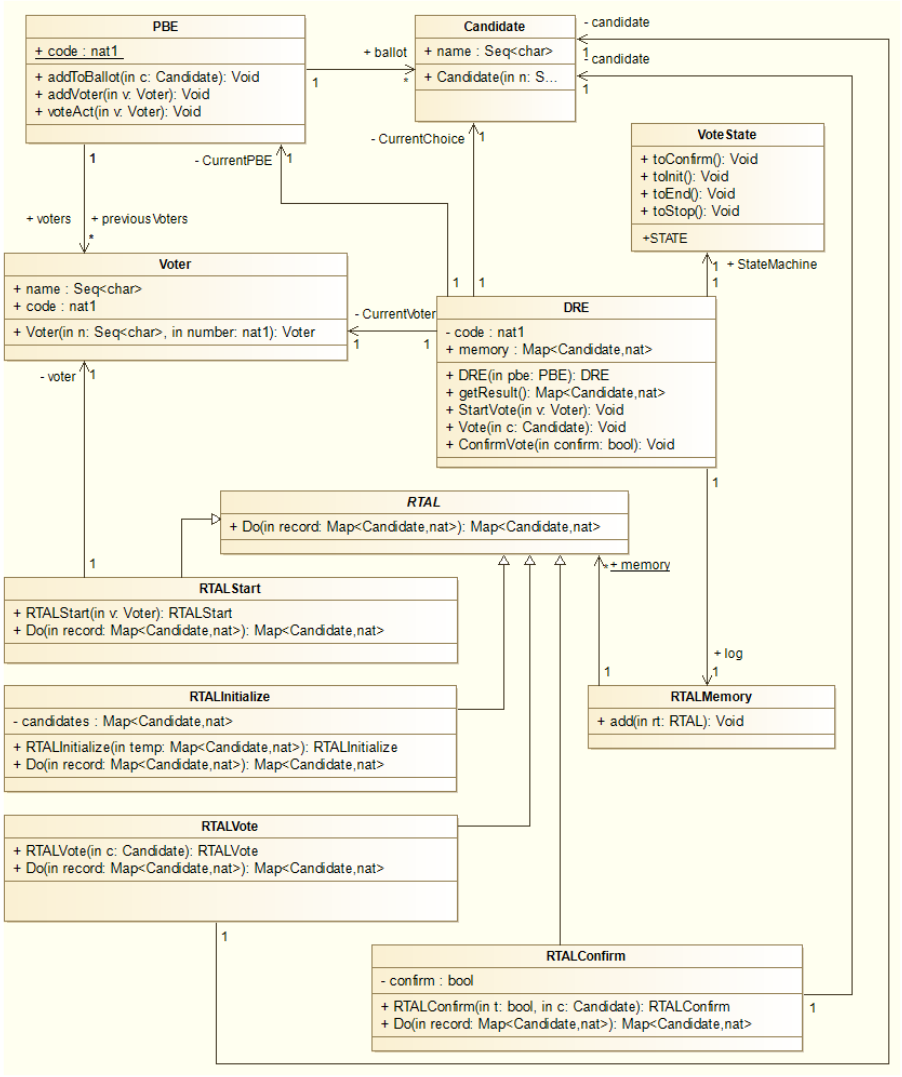

Figure 2: Class Model Diagram.

Function or operation	Line	Coverage	Calls
Candidate	6	100.0%	140
Candidate.vdmpp		100.0%	140

3.2 Voter

```
class Voter
instance variables
 public name: seq of char;
 public code: nat1;
operations
--Constructor-Create a new Voter

 public Voter: seq of char * nat1 ==> Voter
 Voter(n,number) == (name := n;
           code:=number;
           return self)
 pre
```

```
 post c in set ballot;
--Add new Voter to PBE voters set
  public addVoter: Voter ==> ()

 addVoter(v) == voters := {v} union voters
 pre v.name <> []
  and v.code > 1000
  and v.code < 9999
 post v in set voters;

--Voter finished
 public voteAct: Voter ==> ()

 voteAct(v) == (
         voters := remove[Voter](v,voters);
         previousVoters := previousVoters union {v}
        )
 pre v in set voters
 post v in set previousVoters
    and v not in set voters
    and card voters = card voters~-1
    and card previousVoters = card previousVoters~+1;
functions
    public remove[@T](e: @T, s: set of @T) res: set of @T ==

     {x | x in set s & x <> e};
end PBE
```

Function or operation	Line	Coverage	Calls
addToBallot	11	100.0%	140
addVoter	17	100.0%	40
remove	36	100.0%	25
voteAct	25	100.0%	25
PBE.vdmpp		100.0%	230

3.4 VoteState

```
class VoteState
types
 public STATE = <INIT> | <CONFIRM> | <STOP> | <END>;
instance variables
 public currentState: STATE := <STOP>;
operations

 public toConfirm: () ==> ()
 toConfirm() == currentState := <CONFIRM>
 pre currentState = <INIT>
 post currentState = <CONFIRM>;
```

```
 public toInit: () ==> ()
 toInit() == currentState := <INIT>
 pre currentState = <STOP> or  currentState = <CONFIRM>
 post currentState = <INIT>;


 public toEnd: () ==> ()
 toEnd() == currentState := <END>
 pre currentState = <CONFIRM>
 post currentState = <END>;


 public toStop: () ==> ()
 toStop() == currentState := <STOP>
 post currentState = <STOP>;
end VoteState
```

Function or operation	Line	Coverage	Calls
toConfirm	7	100.0%	94
toEnd	17	100.0%	14
toInit	12	100.0%	96
toStop	22	100.0%	106
VoteState.vdmpp		100.0%	310

3.5 DRE

```
class DRE
instance variables
 private code: nat1 :=9999;
 public memory: map Candidate to nat := {|->} ;
 public log: RTALMemory := new RTALMemory();
 private CurrentPBE: PBE;
 public StateMachine: VoteState := new VoteState();
 private CurrentChoice: Candidate := new Candidate();
 private CurrentVoter: Voter := new Voter();
operations
-- Constructor

 public DRE: PBE ==> DRE
 DRE(pbe) == (

       CurrentPBE := pbe;
       StateMachine.toStop();
       memory :={|->};
       for all cand in set CurrentPBE.ballot do
          (
        memory:=memory++{cand|->0};
          );
```

```
          log.add(new RTALInitialize(memory));
       return self)
 pre
  pbe.ballot <> {}
  and pbe.voters <> {}
  and pbe.code = code
 post
  self.CurrentPBE.ballot = pbe.ballot
  and self.CurrentPBE.voters = pbe.voters;


 public getResult:() ==> map Candidate to nat
 getResult()== return memory;

--Voting sequence state machine

 public StartVote: Voter ==> ()
 StartVote(v) == (
   log.add(new RTALStart(v));
   StateMachine.toInit();
   CurrentVoter := v;
 )
 pre StateMachine.currentState=<STOP>
   and card CurrentPBE.voters <> 0
   and v in set CurrentPBE.voters
   and v not in set CurrentPBE.previousVoters
 post StateMachine.currentState=<INIT>
    and v = CurrentVoter;


 public Vote: Candidate ==> ()
 Vote(c) == (
   log.add(new RTALVote(c));
   StateMachine.toConfirm();
   CurrentChoice := c
 )
 pre StateMachine.currentState=<INIT>
   and CurrentVoter in set CurrentPBE.voters
   and c in set CurrentPBE.ballot
 post StateMachine.currentState=<CONFIRM>
    and c = CurrentChoice
    and CurrentVoter in set CurrentPBE.voters
    and CurrentChoice in set CurrentPBE.ballot;
 ----------------------------------------------------------------------------------

 public ConfirmVote: bool ==>()
 ConfirmVote(confirm) == (
   log.add(new RTALConfirm(confirm,CurrentChoice));
   if(confirm)
   then(
     memory(CurrentChoice):=memory(CurrentChoice)+1;
     CurrentPBE.voteAct(CurrentVoter);
     if(CurrentPBE.voters = {})
     then(
      StateMachine.toEnd();
     )
     else(
```

```
      StateMachine.toStop();
     );
     )
   else(
     StateMachine.toInit();
     )
 )
 pre StateMachine.currentState=<CONFIRM>
   and CurrentVoter.name <> []
   and CurrentChoice.name <> []
 post (StateMachine.currentState=<STOP>
    and CurrentVoter in set CurrentPBE.previousVoters
    and CurrentVoter not in set CurrentPBE.voters
    )
    or (
    StateMachine.currentState=<INIT>
    and CurrentVoter in set CurrentPBE.voters)
    or (
    StateMachine.currentState=<END>
    and CurrentVoter in set CurrentPBE.previousVoters
    and CurrentVoter not in set CurrentPBE.voters
    );
functions

end DRE
```

Function or operation	Line	Coverage	Calls
ConfirmVote	63	100.0%	74
DRE	12	100.0%	35
StartVote	36	100.0%	68
Vote	49	100.0%	76
getResult	32	100.0%	18
DRE.vdmpp		100.0%	271

3.6 RTAL

```
class RTALMemory
instance variables
  public static memory: seq of RTAL := [];
operations
 public add: RTAL ==> ()

 add(rt)==(
  memory := memory ^ [rt];
 )
 post len memory = len memory~+1;
end RTALMemory
------------------------------------------------------------------------
```

```
class RTAL
instance variables
operations
 public Do : (map Candidate to nat) ==> (map Candidate to nat)
 Do(record) == is subclass responsibility;
end RTAL
------------------------------------------------------------------------
class RTALInitialize is subclass of RTAL
instance variables
 candidates : map Candidate to nat;
operations
 public RTALInitialize: map Candidate to nat ==> RTALInitialize

 RTALInitialize(temp) == (
       candidates := temp;
       return self)
 pre card dom temp <> 0
 post candidates = temp;

 public Do :  (map Candidate to nat) ==> (map Candidate to nat)
 Do(record) ==  (



  dcl temp : map Candidate to nat := record;
  temp :=candidates;
  return temp)
 pre card dom candidates <> 0;

end RTALInitialize



------------------------------------------------------------------------

class RTALStart is subclass of RTAL
instance variables
 voter : Voter;
operations
 public RTALStart: Voter ==> RTALStart
 RTALStart(v) == (
       voter := v;
       return self)

 pre v.name <> []
 post voter = v;

 public Do :  (map Candidate to nat) ==> (map Candidate to nat)
 Do(record) ==  (return record;)
 pre voter.name <> [];


end RTALStart
------------------------------------------------------------
class RTALVote is subclass of RTAL
instance variables
  candidate : Candidate;
```

```
operations
 public RTALVote: Candidate ==> RTALVote
 RTALVote(c) == (
       candidate := c;
       return self);


 public Do :  (map Candidate to nat) ==> (map Candidate to nat)
 Do(record) == (return record;)
 pre candidate.name <> [];
end RTALVote
------------------------------------------------------------------
class RTALConfirm is subclass of RTAL

instance variables
 confirm : bool := false;
 candidate : Candidate;
operations
 public RTALConfirm: bool*Candidate ==> RTALConfirm
 RTALConfirm(t,c) == (
       confirm := t;
       candidate:= c;
       return self)
 post confirm = t;

 public Do : (map Candidate to nat) ==> (map Candidate to nat)
 Do(record) == (
  dcl temp:map Candidate to nat := record;
  if(confirm)
  then(
   temp(candidate):=temp(candidate)+1;
   return temp;
  )
  else(
   return temp;
  );
 )
 pre (confirm = true or confirm = false)
end RTALConfirm
```

Function or operation	Line	Coverage	Calls
Do	73	100.0%	37
Do	56	100.0%	37
Do	39	100.0%	13
Do	39	100.0%	52
Do	24	100.0%	5
RTALConfirm	66	100.0%	20
RTALInitialize	32	100.0%	10
RTALMemory	12	100.0%	5
RTALStart	32	100.0%	13

RTALStart	32	100.0%	18
RTALVote	49	100.0%	20
add	6	100.0%	91
RTAL.vdmpp		100.0%	321

4 Model Validation

4.1 EVSTestSuit

```
class EVSTestSuit

instance variables
 voteTable: PBE := new PBE();
 voterOne: Voter := new Voter("Joaquim",1001);
 voterTwo: Voter :=  new Voter("Joao",1002);
 voterThree: Voter:= new Voter("Correia",1003);
 voterFour: Voter:= new Voter("Rui",1004);
 candidateOne: Candidate:= new Candidate("PS");
 candidateTwo: Candidate:= new Candidate("PSD");
 candidateThree: Candidate:= new Candidate("CDS");
 candidateFour: Candidate:= new Candidate("Livre");
 VotingProcess: DRE := new DRE();
operations


private assertTrue: bool ==> ()
 assertTrue(cond) == return
 pre cond;


private testVoting: () ==> ()
testVoting() ==
(
 dcl record: map Candidate to nat :={|->};

 voteTable.addVoter(voterOne);
 voteTable.addVoter(voterTwo);
 voteTable.addVoter(voterThree);
 voteTable.addVoter(voterFour);
 assertTrue(card voteTable.voters = 4);
 voteTable.addToBallot(candidateOne);
 voteTable.addToBallot(candidateTwo);
 voteTable.addToBallot(candidateThree);
 voteTable.addToBallot(candidateFour);
 assertTrue(card voteTable.ballot = 4);
 VotingProcess:= new DRE(voteTable);
 VotingProcess.StartVote(voterOne);
 VotingProcess.Vote(candidateOne);
 VotingProcess.ConfirmVote(true);
 assertTrue(card voteTable.voters = 3);
 assertTrue(card voteTable.previousVoters = 1);
 VotingProcess.StartVote(voterTwo);
```

```
 VotingProcess.Vote(candidateOne);
 VotingProcess.ConfirmVote(true);
 assertTrue(VotingProcess.getResult()(candidateOne)=2);
 assertTrue(card voteTable.voters = 2);
 assertTrue(card voteTable.previousVoters = 2);

 for entry in RTALMemory`memory
  do (
    record:=entry.Do(record);
  );
  assertTrue(VotingProcess.memory = record);

);


private testReChoiceVoting: () ==> ()
testReChoiceVoting() ==
(
 dcl record: map Candidate to nat:={|->};

 voteTable.addVoter(voterOne);
 voteTable.addVoter(voterTwo);
 voteTable.addVoter(voterThree);
 voteTable.addVoter(voterFour);
 assertTrue(card voteTable.voters = 4);
 voteTable.addToBallot(candidateOne);
 voteTable.addToBallot(candidateTwo);
 voteTable.addToBallot(candidateThree);
 voteTable.addToBallot(candidateFour);
 assertTrue(card voteTable.ballot = 4);
 VotingProcess:= new DRE(voteTable);
 VotingProcess.StartVote(voterOne);
 VotingProcess.Vote(candidateOne);
 VotingProcess.ConfirmVote(false);
 assertTrue(card voteTable.voters = 4);
 assertTrue(card voteTable.previousVoters = 0);
 VotingProcess.Vote(candidateTwo);
 VotingProcess.ConfirmVote(true);
 assertTrue(VotingProcess.getResult()(candidateTwo)=1);
 assertTrue(card voteTable.voters = 3);
 assertTrue(card voteTable.previousVoters = 1);

 for entry in RTALMemory`memory
  do (
    record:=entry.Do(record);
  );


  assertTrue(VotingProcess.memory = record);
);

private testEnd: () ==> ()
testEnd() ==
(
 dcl record: map Candidate to nat:={|->};
 voteTable.addVoter(voterOne);
 voteTable.addVoter(voterTwo);
```

```
 voteTable.addVoter(voterThree);
 voteTable.addVoter(voterFour);
 assertTrue(card voteTable.voters = 4);
 voteTable.addToBallot(candidateOne);
 voteTable.addToBallot(candidateTwo);
 voteTable.addToBallot(candidateThree);
 voteTable.addToBallot(candidateFour);
 assertTrue(card voteTable.ballot = 4);
 VotingProcess:= new DRE(voteTable);
 VotingProcess.StartVote(voterOne);
 VotingProcess.Vote(candidateOne);
 VotingProcess.ConfirmVote(true);
 VotingProcess.StartVote(voterTwo);
 VotingProcess.Vote(candidateOne);
 VotingProcess.ConfirmVote(true);
 VotingProcess.StartVote(voterThree);
 VotingProcess.Vote(candidateTwo);
 VotingProcess.ConfirmVote(true);
 VotingProcess.StartVote(voterFour);
 VotingProcess.Vote(candidateTwo);
 VotingProcess.ConfirmVote(true);
 assertTrue(VotingProcess.StateMachine.currentState = <END>);

 --audit
 for entry in RTALMemory`memory
  do (
    record:=entry.Do(record);
  );


  assertTrue(VotingProcess.memory = record);
);

public static main: () ==> ()
main() ==(
 new EVSTestSuit().testEnd();
 new EVSTestSuit().testVoting();
 new EVSTestSuit().testReChoiceVoting();
);

end EVSTestSuit
```

Function or operation	Line	Coverage	Calls
assertTrue	16	100.0%	200
main	125	100.0%	4
testEnd	88	100.0%	4
testReChoiceVoting	55	100.0%	3
testVoting	20	100.0%	9
EVSTestSuit.vdmpp		100.0%	220

5 Bibliography

[1] Pipattanasomporn M., Feroze H. and Rahman S., Multi-agent systems in a distributed smart grid: Design and implementation, 2009.