Electronic Voting System

Assignment 5 (T05)

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 15, 2014

Contents

1 Informal system description and list of requirements

1.1 Informal system description

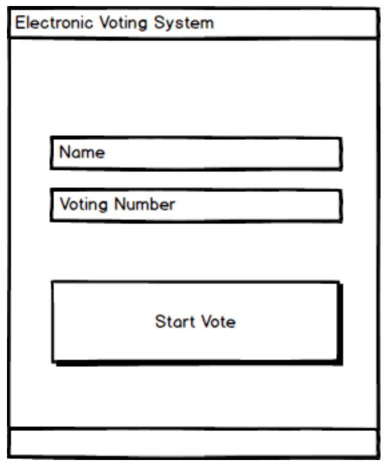

Figure 1: A voter can enter the credentials to vote.

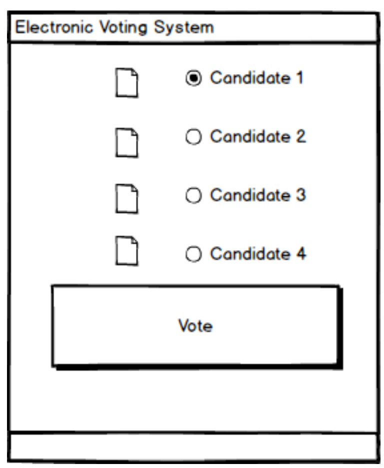

Figure 2: A voter can choose the candidate.

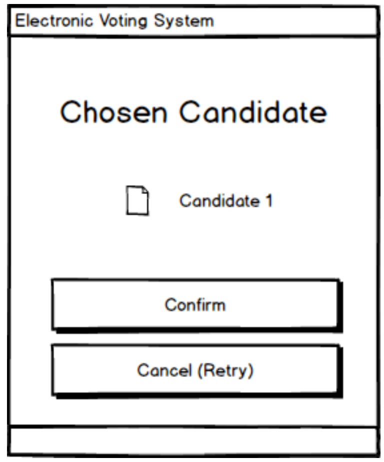

Figure 3: Final window. Confirm vote or cancel and repeat the choice action.

1.2 Requirements List

ID	Priority	Description
R1	Mandatory	The Voter can start voting by indicating his name and code.
R2	Mandatory	The Voter can choose candidate from ballot by indicating his name.
R3	Mandatory	The Vote Table can create/manage an election adding it a new Voter or a new Candidate.
R4	Mandatory	The Vote Table, after the voting action from Voter, should be able to remove from the list that Voter, using his name and code.
R5	Mandatory	The system has to give the possibility to run audits on the elections.

2 Visual UML Model

2.1 Use case model

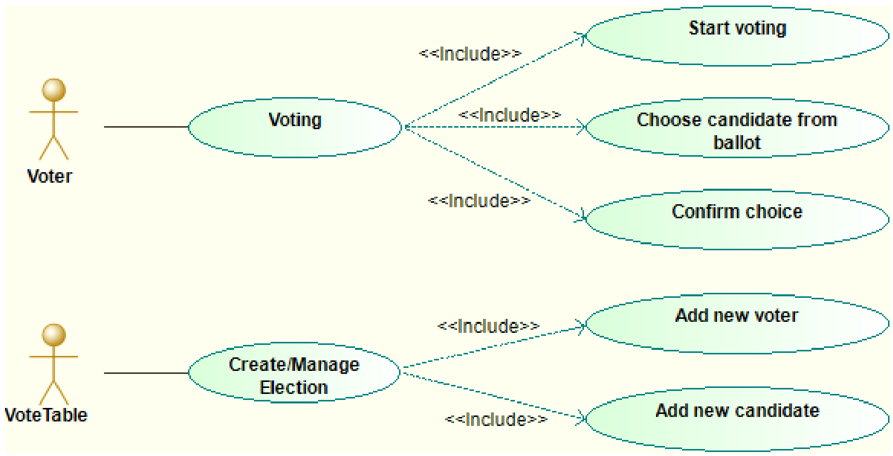

Figure 4: Use Case Diagram.

Scenario	**Starting Vote**
Description	Normal scenario for initialize the vote act and for update the current voter.
Pre-conditions	1.The current state of State Machine is STOP. 2.The voters of the PBE are not empty. 3.The current voter is in voters of PBE. (*input*) 4.The current voter is not in voters of PBE. (*input*)
Post-conditions	1. The current State of State Machine is INIT. (*final system state*) 2. The Voter is the same voter in the ballot. (*final system state*)
Steps	(unspecified)
Exceptions	(unspecified)

Scenario	**Choose candidate from ballot**
Description	Normal scenario for choosing a candidate from a list of them.
Pre-conditions	1.The current state of State Machine is INIT. 2.The current voter is in PBE voters. 3.The candidate exists in the ballot. (*input*)
Post-conditions	1.The current State of State Machine is CONFIRM. (*final system state*) 2.The voted candidate is the same that the selected in ballot. (*final system state*) 3.The current voter is in PBE voters. (*final system state*) 4.The current candidate is in PBE ballot. (*final system state*)
Steps	(unspecified)
Exceptions	(unspecified)

Scenario	**Confirm Vote**
Description	Normal scenario for choosing a candidate from a list of them.
Pre-conditions	1.The current state of State Machine is confirm. 2.The Voter name exists. 3.The Candidate name exists.
Post-conditions	1.The current State of State Machine is STOP. (*final system state*) 2.The current voter is in PBE previous voters. (*final system state*) 3.The current voter is not in PBE voters. (*final system state*) Or 1.The current State of State Machine is INIT. (*final system state*) 2.The current voter is in PBE voters. (*final system state*) Or 1.The current State of State Machine is END. (*final system state*) 2.The current voter is in PBE previous voters. (*final system state*) 3.The current voter is not in PBE voters. (*final system state*)
Steps	1.The Voter chooses a candidate. 2.The Voter confirm is vote.
Exceptions	(unspecified)

Scenario	**Add new Voter**
Description	Normal scenario for adding a Voter to PBE.
Pre-conditions	1.The Voter name is not empty. (*input*) 2.The Voter code is valid (between 1000 and 9999).(*input*)
Post-conditions	1.The Voter is added to PBE as intended. (*final system state*)
Steps	(unspecified)
Exceptions	(unspecified)

Scenario	Add new Candidate
Description	Normal scenario for adding a Candidate to PBE.
Pre-conditions	1.The Candidate name is not empty.(*input*)
Post-conditions	1.The Candidate is added to PBE as intended. (*final system state*)
Steps	(unspecified)
Exceptions	(unspecified)

2.2 State machine model

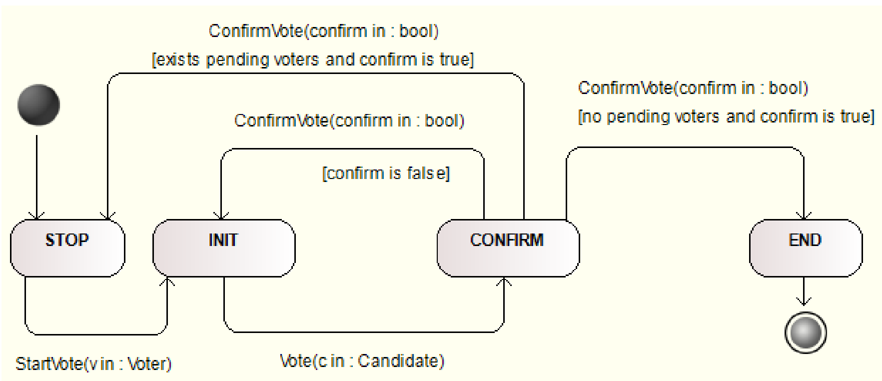

Figure 5: State Machine Diagram (Vote Process).

2.3 Class model

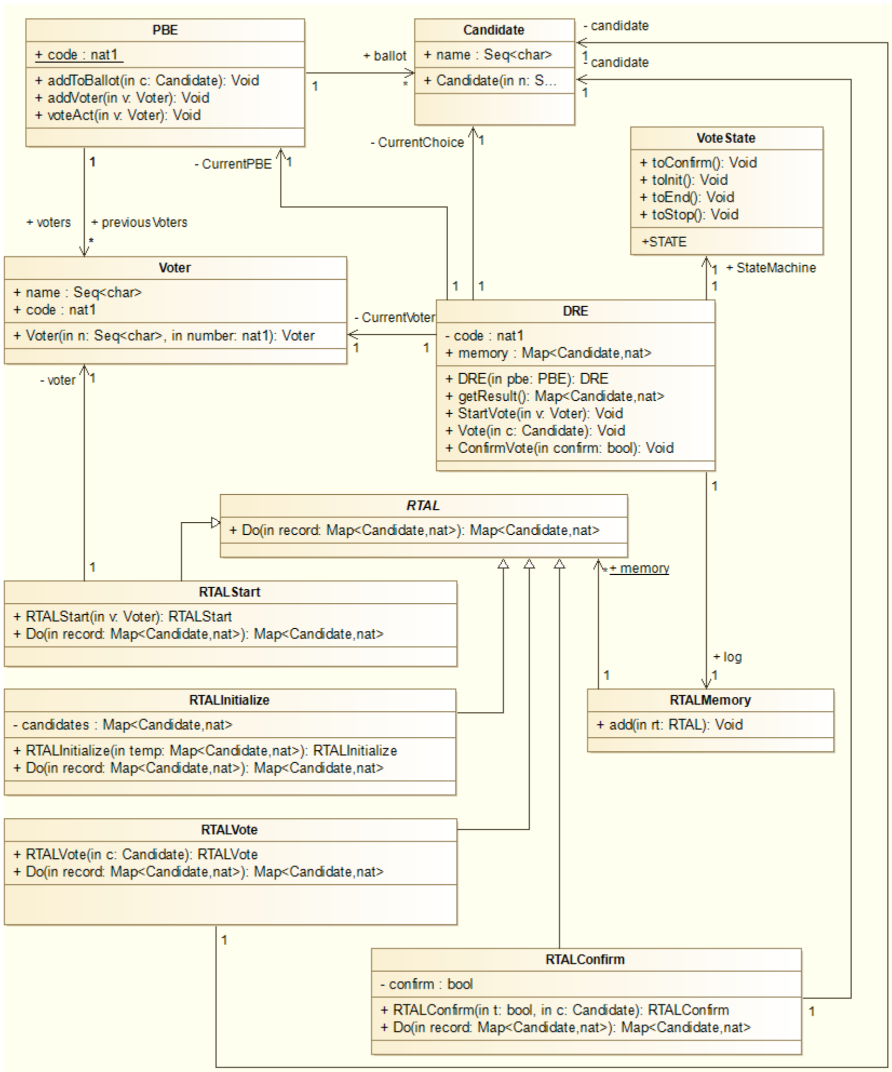

Figure 6: Class Model Diagram.

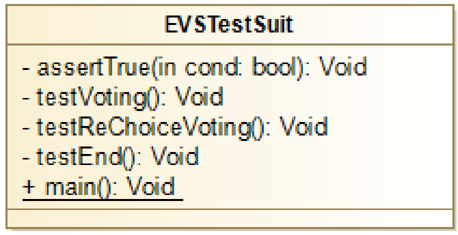

Figure 7: Class Model Diagram.

Class	**Description**
Candidate	Defines a Candidate present in the election ballot.
Voter	Defines a Voter present in the voters list for the election.
PBE	Defines the vote table that contains all the voters and defines the election ballot
DRE	Core Model, defines the state variables and operations available to the voters.
VoteState	Define the set of possible states and the operations to change state.
RTAL	Superclass for RTAL classes; defines RTALInitialize, RTALStart, RTALVote and RTALConfirm. Used to save each action in DRE by the voters.
RTALMemory	Defines the system memory using RTAL objects.

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
```

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
  name <> [] and
  number >= 1000 and
  number <= 9999
 post
  name = n and
  code = number
end Voter
```

Function or operation	Line	Coverage	Calls
Voter	7	100.0%	140
Voter.vdmpp		100.0%	140

3.3 PBE

```
class PBE
instance variables
 public ballot: set of Candidate := {};
 public voters: set of Voter := {};
 public previousVoters:  set of Voter :={};
 public static code: nat1 :=9999;
 --inv cannot exists the same vouter in previousVoters and vice-versa
 inv card (voters inter previousVoters) = 0;
operations
--Add new Candidate to PBE ballot set

 public addToBallot: Candidate ==> ()
 addToBallot(c) == ballot := {c} union ballot
 pre c.name <> []
 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 Model Verification

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
5	DRE‘ConfirmVote(bool)	legal map application

The code under analysis is the presented Proof Obligation:

```
(forall confirm:bool & (((StateMachine.currentState) = <CONFIRM>)
and (((CurrentVoter.name) <> []) and ((CurrentChoice.name) <> [])))
 => (CurrentChoice in set (dom memory))))
```

In this case the proof is trivial because the quantification ’CurrentChoice in set (dom memory)’ assures that the map is accesses only inside its domain.

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	PO Name	Type
37	PBE‘voteAct(Voter)	state invariant holds

The code under analysis is:

```
public voteAct: Voter ==> ()
voteAct(v) == (
      voters := remove[Voter](v,voters);
        previousVoters := previousVoters union {v}
      )
```

The relevant invariant under analysis is:

```
inv card (voters inter previousVoters) = 0;
```

After the execution of the code block under analysis we have (technically, this is the post-condition of the block):

```
v in set previousVoters
and v not in set voters
and card voters = card voters~-1
and card previousVoters = card previousVoters~+1;
```

We have to prove that this implies that the invariant holds, i.e., that the following condition holds:

```
(forall v:Voter & ((v in set voters) =>
(((card (voters inter previousVoters)) = 0) =>
((card ((remove)[Voter](v, voters) inter previousVoters)) = 0))))
```

This formally implies that:

```
(v in set voters) => (card ({v} inter previousVoters) = 0)
```

which is obviously true since if we have a pending 'voter' it cannot be in 'previousVoters'.

6 Conclusions

The model that was developed covers all the requirements as presented in *Formal Specification and Analysis of an E-voting System*[1] and *Formal analysis of an electronic voting system: An experience report*[2].

With the objective of adding another layer of verifications to the system, it has been added to the requirements in [1] and [2], an auditing system, capable of verify if the results of an election are valid. This replicates a well-known pattern called *Command pattern*[1].

This project took approximately 22 hours to develop.

7 References

[1] Formal Specification and Analysis of an E-voting System, ARES 2010, Fifth International Conference on Availability, Reliability and Security, 15-18 February 2010, Krakow, Poland

[2] Formal analysis of an electronic voting system: An experience report, Weldemariam, K., et al., J. Syst. Software (2011)

[3] VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014

[4] Overture tool web site, http://overturetool.org

[5] Command Pattern, Object Oriented Design, http://www.oodesign.com/command-pattern.html, 2014.

[1] Command design pattern encapsulates commands (method calls) in objects allowing us to issue requests without knowing the requested operation or the requesting object. Command design pattern provides the options to queue commands, undo/redo actions and other manipulations[5].