

MAT 7003 : Mathematical Foundations

(for Software Engineering)

J Paul Gibson, A207

`paul.gibson@it-sudparis.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/MAT7003/>

Evoting Problem: Sample Solution

[.../~gibson/Teaching/MAT7003/L2-3-Evoting-SampleSolution.pdf](http://www-public.it-sudparis.eu/~gibson/Teaching/MAT7003/L2-3-Evoting-SampleSolution.pdf)

An Electronic Voting Problem – step 1: a referendum

Do you agree that ...

YES X NO

CONTEXT

Referendum // *To model a referendum where there is a single yes/no response*

SETS

RESPONSE // *The set of choices offered to the VOTERS during the vote*

VOTERS // *The finite set of people entitled to record a vote*

CONSTANTS

yes // *first possible response*

no // *second possible response*

AllVotes // *Linking voters to responses*

CountYes // *The integer count of the yes votes*

CountNo // *The integer count of the no votes*

v1 // *Voter1 - for testing axioms*

v2 // *Voter2 - for testing axioms*

Note: I have made minor changes to the specification given in the previous lecture, in order to shorten the code/presentation and improve the comments

An Electronic Voting Problem – step 1: a referendum

Do you agree that ...

YES X NO

AXIOMS

```
axm1 : (RESPONSE = {yes} u {no}) ∧ // In a referendum there are only 2 possible responses
      yes ≠ no

axm2 : AllVotes ∈ VOTERS ↔ RESPONSE // VOTERS may or may not record a vote,
      // but if they do then they record only one

axm3 : fi nite(VOTERS) ∧ // The number of VOTERS is finite and there is at least one of them
      card (VOTERS) > 0

axm4 : CountYes = card (AllVotes ▷ {yes}) // The number of yes votes
axm5 : CountNo = card (AllVotes ▷ {no}) // The number of no votes

thm_test1 : (VOTERS = {v1, v2} ∧ AllVotes = { v1↦ yes } )⇒ // A zero no vote count check
            CountNo = 0

thm_test2 : (VOTERS = {v1, v2} ∧ AllVotes = { v1↦ yes, v2↦ no } ) ⇒ // A single yes vote count check
            CountYes = 1
```

TO DO: Download the zipped directory `Event-B-VotingMachine.zip` and import the Referendum context (and the other E-voting machine specifications) into RODIN.

An Electronic Voting Problem – step 1: a referendum

An Event-B Specification of Referendum
Creation Date: 25 Oct 2010 @ 01:51:58 PM

CONTEXT Referendum

To model a referendum where there is a single yes/no response

SETS

RESPONSE The set of choices offered to the VOTERS during the vote

VOTERS The finite set of people entitled to record a vote

CONSTANTS

yes first possible response

no second possible response

AllVotes Linking voters to responses

CountYes The integer count of the yes votes

CountNo The integer count of the no votes

v1 Voter1 - for testing axioms

v2 Voter2 - for testing axioms

AXIOMS

axm1 : $(RESPONSE = \{yes\} \cup \{no\}) \wedge$
 $yes \neq no$

In a referendum there are only 2 possible responses

axm2 : $AllVotes \in VOTERS \leftrightarrow RESPONSE$

VOTERS may or may not record a vote, but if they do then they record only one

axm3 : $finite(VOTERS) \wedge$
 $card(VOTERS) > 0$

The number of VOTERS is finite and there is at least one of them

axm4 : $CountYes = card(AllVotes \triangleright \{yes\})$

The number of yes votes

axm5 : $CountNo = card(AllVotes \triangleright \{no\})$

The number of no votes

thm_test1 : $(VOTERS = \{v1, v2\} \wedge AllVotes = \{v1 \mapsto yes\}) \Rightarrow$
 $CountNo = 0$

A zero no vote count check

thm_test2 : $(VOTERS = \{v1, v2\} \wedge AllVotes = \{v1 \mapsto yes, v2 \mapsto no\}) \Rightarrow$
 $CountYes = 1$

A single yes vote count check

END

**The specification file -
Referendum_25_Oct_2010.pdf
- should be cross-referenced in
any subsequent documentation**

**This pdf was
generated using the
B2Latex RODIN
plugin, (followed by
mikTex)**

An Electronic Voting Problem : a referendum

Implementing the RESPONSE set as a Java class

With good documentation, including **cross-references with the Event-B specification**, you should not need to look at the code:

© Response

Version:

1

A simple class for specifying the allowed responses to a referendum

Implements the RESPONSE set in the Event-B context [Referendum](#) :

$axm1 : (RESPONSE = \{yes\} \cup \{no\}) \wedge yes \neq no // In a referendum there are only 2 possible responses$

Will be re-used in the [Referendum](#) class and its test class [ReferendumTest](#)

[Referendum_25_Oct_2010.pdf](#)

[Referendum.html](#)

The main design decision was to wrap an enumeration in a class. Then a voter abstaining would be a NULL Response object.

Author:

J Paul Gibson (paul.gibson@it-sudparis.eu)

The html documentation view generated by Javadoc

An Electronic Voting Problem : a referendum

The Javadoc Source (comments) for the RESPONSE class :

```
/**
 *
 * @author J Paul Gibson
 * (paul.gibson@it-sudparis.eu)
 * @version 1
 * <br>
 * A simple class for specifying the allowed responses to a referendum <br>
 * Implements the RESPONSE set in the Event-B context
 * Referendum :
 * <em><ul>
 * axml      :      (RESPONSE = {yes} U {no})   $\wedge$    yes  $\neq$  no
 * //      In a referendum there are only 2 possible responses <br>
 * </ul></em>
 * Will be re-used in the {@link Referendum} class
 * and its test class {@link ReferendumTest} <br><br>
 * The main design decision was to wrap an enumeration in a class.
 * Then a voter abstaining would be a NULL Response object.
 **/
```

An Electronic Voting Problem : a referendum

Now, let us write our test class based on the Event-B specification

```
thm_test1 : (VOTERS = {v1, v2} ^ AllVotes = { v1↦ yes } )⇒ // A zero no vote count check
            CountNo = 0
thm_test2 : (VOTERS = {v1, v2} ^ AllVotes = { v1↦ yes, v2↦ no }) ⇒ // A single yes vote count check
            CountYes = 1
```

As these tests are not very comprehensive, we will add another test which generates (randomly seeded) a Referendum vote and shows the calculated counts.

NOTE: We would normally use a test tool (like JUnit) to support the testing process, but for this simple example we will code them ourselves without specific test tool support.

An Electronic Voting Problem –: a referendum

Expected output on specified tests:

```
***** Test thm_test1 of specification *****
```

```
Referendum.allVotes ---
```

```
1: yes
```

```
Count yes = 1
```

```
Count no = 1
```

```
Count abstain = 1
```

```
Checking CountNo = 0 is true
```

```
***** Test thm_test2 of specification *****
```

```
Referendum.allVotes ---
```

```
1: yes
```

```
2: no
```

```
Count yes = 1
```

```
Count no = 1
```

```
Count abstain = 0
```

```
Checking CountYes = 1 is true
```

An Electronic Voting Problem –: a referendum

Expected typical output on random test:

```
***** Test a random referendum *****  
Referendum.allVotes ---  
1: no  
4: yes  
5: yes  
6: no  
7: no  
10: yes  
  
Count yes = 3  
Count no = 3  
Count abstain = 4
```

We combine these test to produce a single test file (and we must also provide information concerning when the test was run and the random seed that can be used to reproduce the test)

An Electronic Voting Problem –: a referendum

EXPECTED OUTPUT (when the default seed value 0 is used):

The seed used for the random number generator in the test is 0.

You can override this value by passing an integer value as a main argument parameter, if you so wish.

```
*** Referendum Test 0 Execution Date/Time 2010/10/22 16:30:00 ***
```

```
***** Test thm_test1 of specification *****
```

```
...
```

```
***** Test thm_test2 of specification *****
```

```
...
```

```
***** Test a random referendum *****
```

An Electronic Voting Problem –: a referendum

The Referendum test class:

ReferendumTest

Version:

1

Simple Test For [Referendum](#) class, based on the Event-B Specification of context [Referendum](#)

(NOTE: In this version we do not use the JUnit testing tool because the students have yet to see how to use it.)

Tests the following behaviour:

- thm_test1 of specification Referendum: With a single yes vote, the no count is zero;
- thm_test2 of specification Referendum: With a single yes vote and a single no vote, the yes count is one;
- Initialise a Random Referendum (Between 10 and 20 Voters), using a default seed value of 0 if the command line does not include a positive integer seed as a first parameter;
- Print out the random Referendum state to visually check that the result is as expected;

EXPECTED OUTPUT (when the default seed value 0 is used):

Expected Output -As on previous slide

An Electronic Voting Problem –: a referendum

The Referendum test class (code for each specified test):

```
Referendum referendum;

referendum = new Referendum(2);

// Test thm_test1 of specification
System.out.println("\n***** Test thm_test1 of specification *****\n");
referendum.addVote(1, Response.YES);
System.out.println(referendum);
System.out.println("Checking CountNo = 0 is "+ (referendum.countNo()==0));

// Test thm_test2 of specification
System.out.println("\n***** Test thm_test2 of specification *****\n");
referendum.addVote(2, Response.NO);
System.out.println(referendum);
System.out.println("Checking CountYes = 1 is "+ (referendum.countYes()==1));
```

An Electronic Voting Problem –: a referendum

The Referendum test class (code for the random test):

```
// Additional Random Test

System.out.println("\n***** Test a random referendum *****\n");
int numberOfVoters = rng.nextInt(11)+10;

referendum = new Referendum(numberOfVoters);

for (int voteCount=1; voteCount<=numberOfVoters; voteCount++){
    if (rng.nextInt(3)==0) referendum.addVote(voteCount, Response.YES);
        else if (rng.nextInt(2)==0)referendum.addVote(voteCount,
Response.NO);
}
System.out.println(referendum);
```

An Electronic Voting Problem : a referendum

All Classes

[Referendum](#)
[ReferendumTest](#)
[Response](#)
[Response.RESPONSE](#)

Package **Class** Use Tree Deprecated Index Help

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class Referendum

java.lang.Object
└ **Referendum**

```
public class Referendum  
extends java.lang.Object
```

Version:

1

Implements the Event-B [Referendum](#) Context Specification

Re-uses the [Response](#) class to record *yes/no* votes

The main design decision were to;

- implement the set of *VOTERS* by the subset of integers $\{1, .. numberOfVoters\}$
- implement *AllVotes* as an array [allVotes](#) [0.. (numberOfVotes-1)] of [Responses](#), where a NULL entry corresponds to the voter not having voted.
- implement *CountYes* and *CountNo* as [countYes\(\)](#) and [countNo\(\)](#) methods of the [Referendum](#) class

Author:

J Paul Gibson (paul.gibson@it-sudparis.eu)

TO DO:

- download `Java-Evoting (MAT7003) .zip`,
- compile and execute the Java Referendum test code
- compare the documentation with the Java code and the Event-B

An Electronic Voting Problem : a presidential election

Presidential_25_Oct_2010.pdf

An Event-B Specification of Presidential
Creation Date: 25 Oct 2010 @ 03:56:28 PM

CONTEXT Presidential

In a presidential election, each voter who chooses to vote must select a single candidate

SETS

VOTERS The finite set of people entitled to vote in the election, but who do not necessarily do so

CONSTANTS

AllVotes A function linking each voter, who recorded a vote, to their selected candidate

CountVote A function linking each candidate to the number of votes they have received

Candidates The subset of voters who run as candidates in the election

v1 Voter for testing

v2 Voter for testing

AXIOMS

axm1 : $Candidates \subseteq VOTERS$

axm2 : $AllVotes \in VOTERS \leftrightarrow Candidates$

axm3 : $finite(VOTERS) \wedge card(VOTERS) > 0$

axm4 : $CountVote \in Candidates \rightarrow \mathbb{N}$

axm5 : $\forall c. c \in Candidates \Rightarrow$
 $CountVote(c) = card(AllVotes \triangleright \{c\})$

axm6 : $v1 \in VOTERS$

axm7 : $v2 \in VOTERS$

axm8 : $v1 \neq v2$

axm9 : $AllVotes = \{v1 \mapsto v2, v2 \mapsto v2\} \Rightarrow CountVote(v2) = 2$

END

An Electronic Voting Problem : a presidential election

Presidential.html

All Classes

[Presidential](#)
[PresidentialTest](#)
[Referendum](#)
[ReferendumTest](#)
[Response](#)
[Response.RESPONSE](#)

Package **Class** Use Tree Deprecated Index Help

PREV CLASS [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Class Presidential

java.lang.Object
└ **Presidential**

```
public class Presidential  
extends java.lang.Object
```

Version:

1
Implements the Event-B [Presidential](#) Context Specification

The main design decision were to;

- implement the set of *VOTERS* by the subset of integers $\{1, .. numberOfVoters\}$
- implement the set of *CANDIDATES* as a boolean array isCandidate for each voter
- implement *AllVotes* as an array [Referendum.allVotes](#) $[0.. (numberOfVoters-1)]$ of integers, where an entry corresponding to a valid candidate is a vote for that candidate, and otherwise and invalid candidate entry represents a voter who has not voted .
- implement *CountVote* as the [countVote\(int\)](#) method

Author:

J Paul Gibson (paul.gibson@it-sudparis.eu)

An Electronic Voting Problem : a presidential election

```
*****  
Presidential Test 0  
Execution Date/Time 2010/10/25 17:13:31  
*****
```

```
***** Test axm9 of specification *****
```

```
Presidential.allVotes ---  
Votes Recorded ---  
1: 2  
2: 2  
Candidate Counts ---  
2: 2  
  
Checking CountVote(2) = 2 is true
```

An Electronic Voting Problem : a presidential election

***** Test a random referendum *****

Presidential.allVotes ---

Votes Recorded ---

1: 10

2: 6

3: 2

4: 0

5: 5

6: 8

7: 3

8: 0

9: 5

10: 0

Candidate Counts ---

2: 1

3: 1

5: 2

6: 1

8: 1

10: 1

TO DO:

- download `Java-Evoting(MAT7003).zip`,
- compile and execute the Presidential Java test code
- compare the documentation with the Java code and the Event-B specification

An Electronic Voting Problem : a list election

An Event-B Specification of ListElection
Creation Date: 25 Oct 2010 @ 05:29:02 PM

CONTEXT ListElection

An election where we can list all the candidates that we support

SETS

VOTERS

CONSTANTS

Candidates A subset of VOTERS

ValidVotes All valid (non-empty) candidate lists which follow the minimum and maximum requirements

v1

v2

v3

minimum

maximum

UnrestrictedValidVotes No restriction from minimum or maximum requirements

AXIOMS

axm1 : $finite(VOTERS)$

axm2 : $Candidates \subseteq VOTERS$

axm3 : $UnrestrictedValidVotes = \mathbb{P}(Candidates) \setminus \emptyset$

axm4 : $minimum \in 1 .. card(Candidates)$

axm5 : $maximum \in 1 .. card(Candidates)$

axm6 : $minimum \leq maximum$

axm7 : $ValidVotes \subseteq UnrestrictedValidVotes$

axm9 : $v1 \in VOTERS$

axm10 : $v2 \in VOTERS$

axm11 : $v3 \in VOTERS$

axm12 : $(v1 \neq v2) \wedge (v1 \neq v3) \wedge (v2 \neq v3)$

axm13 : $Candidates = \{v1, v2, v3\} \Rightarrow$
 $\{v1, v3\} \in UnrestrictedValidVotes$

END

TO DO:
Implement the list election and test and document your code appropriately

An Electronic Voting Problem : a preferential election

| | |
|--------|---|
| Smith | 2 |
| Jones | |
| Gibson | 1 |
| Park | 3 |
| Green | |

An Event-B Specification of Preferential
Creation Date: 25 Oct 2010 @ 06:09:47 PM

CONTEXT Preferential

CONSTANTS

numCandidates

CANDIDATES

PREFERENCES

singleVote

AXIOMS

axm1 : $numCandidates \in \mathbb{N}_1$

axm2 : $CANDIDATES = 1 .. numCandidates$

axm3 : $PREFERENCES = 0 .. numCandidates$

axm4 : $singleVote \in CANDIDATES \rightarrow PREFERENCES$

axm5 : $\forall i, j. i \in CANDIDATES \wedge j \in CANDIDATES \wedge$
 $singleVote(i) = singleVote(j) \Rightarrow$
 $((i = j) \vee singleVote(i) = 0)$

No repeated non-zero preferences

axm6 : $\exists m. m \in \mathbb{N}_1 \wedge$

$(0 .. m = ran(singleVote))$

No missing elements in preference sequence

END

TO DO: Validate this specification – it may not be correct !!

You can download it from the web site:

[Event-B-VotingMachine.zip](#)

An Electronic Voting Problem– Vote Parcels

An Event-B Specification of VoteParcel
Creation Date: 25 Oct 2010 @ 05:39:24 PM

CONTEXT VoteParcel

A Stack (LIFO) of Votes

SETS

VOTES

PARCEL

CONSTANTS

noVotes An empty Parcel

addVote To any parcel we can add a vote

removeVote We can remove the last vote added to any nonempty Parcel

top We can access/read the vote on top of the Parcel

countVotes The number of votes in a Parcel

transferVote Move a vote from the top of the 1st Parcel to the top of the 2nd Parcel

AXIOMS

axm1 : $noVotes \in PARCEL$

axm2 : $addVote \in (PARCEL \times VOTES) \rightarrow (PARCEL \setminus \emptyset)$

axm3 : $removeVote \in (PARCEL \setminus \emptyset) \rightarrow PARCEL$

axm4 : $\forall p, v. p \in PARCEL \wedge v \in VOTES \Rightarrow$
 $removeVote(addVote(p \mapsto v)) = p$

axm5 : $top \in (PARCEL \setminus \emptyset) \rightarrow VOTES$

axm6 : $\forall p, v. p \in PARCEL \wedge v \in VOTES \Rightarrow$
 $top(addVote(p \mapsto v)) = v$

axm7 : $countVotes \in PARCEL \rightarrow \mathbb{N}$

axm8 : $countVotes(noVotes) = 0$

axm9 : $transferVote \in ((PARCEL \setminus \emptyset) \times PARCEL) \rightarrow (PARCEL \times (PARCEL \setminus \emptyset))$

axm10 : $\forall p1, p2, v. p1 \in PARCEL \wedge p2 \in PARCEL \wedge v \in VOTES \Rightarrow$
 $transferVote((addVote(p1 \mapsto v)) \mapsto p2) =$
 $(p1 \mapsto addVote(p2 \mapsto v))$

END

Electronic Voting: Use Specification to Develop an Implementation

The specification models (as Event-B contexts) can be downloaded from the module web site.

Each specification model should have a corresponding implementation together with test code (developed from the specification) – i have provided some example implementations and tests (in Java). It is for you to complete the unfinished implementations.

You should look at the sample code, tests and documentation in order to see the level of rigour that we expect in all the code that you develop during this module/programme.