Final Year Project Interim Report Deliverable

Implementing the Irish Voting System

Patrick Tierney

Submitted in partial fulfilment of the degree of
BA(Hons.) in the School of Computer Science and
Informatics with the supervision of Dr. Joseph Kiniry
and moderated by Prof. Pádraig Cunningham

School of Computer Science and Informatics

University College Dublin

Table of Contents

1 Project Description 4

2 Project Focus 2

2.1 Key Milestones 2

2.2 Literature Review 5

3 Progress to Date 6

3.1 Specification Faults 8

3.2 Design 9

3.2 Verification 9

4 Future Work 10

5 References 12

6 Appendices 14

A Specification Faults 14

A1 Ballot.java 14

A2 Candidate.java 15

B Design Changes 16

A1 Ballot.java 16

A2 Candidate.java 17

1 Project Description

The aim of this project is to implement the Irish Voting System Law (known as "votáil" in Irish) according to a formal specification using the design by contract methodology. A larger project concerning electronic voting in general incorporates the work developed in this paper. Part of the overall project is creating the specifications needed to implement the Irish Voting System, completed by Dermot Cochran in 2005-2006 [1]. According to these specifications, the Irish Voting System will be implemented using tools as part of the Java Modeling Language (JML) suite and the Extended Static Checker for Java Version 2 (ESC/Java2). JML is a formal specification language for Java [10] while ESC/Java2 is a programming tool that attempts to find common run-time errors in JML-annotated Java programs [6]. The system is unit tested using JML unit tools when all the work is completed.

2 Project Focus

2.1 Key Milestones

Mandatory

- Become familiar with the Irish voting system law.
- Implement, using JML-annotated Java, the Irish voting system according to the pre-existing architecture design and formal specification.

Discretionary

- Write an EBON specification of the system.
- Generate and execute unit tests using JML-jUnit.
- Check the system implementation rigorously with ESC/Java2.
- Contribute this implementation to the Verified Software Repository.

Exceptional

- Write unit tests corresponding to EBON scenarios using jUnit.
- Co-author, submit, and publish a paper on this work.
- Specify and prove some (behavioural and security) properties about the specification.

2.2 Literature Review

The first item that had to be researched was the Irish Voting Law, and how elections in the Republic of Ireland operate [4]. As most of this is all ready taught in Irish secondary schools, it was just a case of revising the subject. Understanding what occurs in exceptional situations, for example, as in the case of two candidates having the exact same number of votes, is important to ensure the correct implementation and testing of the system. The electoral act of 2004 provides the information about these exceptional situations [3].

The JML specification of the Irish Voting System for the project was studied thoroughly. As this plays a major part in this project, a lot of time was spent reading the specification. All though the specifications were easy to understand, there were some problems with some of the invariants in the classes. Thankfully, some notes from Dermot Cochran also helped to interpret these problems.

The JML and ESC/Java2 tools are vitally important in this project, so reading papers and tutorials on both of these areas was essential [5-11]. Other papers that are relevant to the project are referenced in Section 4.

A close eye was also kept on the worldwide elections that occurred during October and November. The elections in the U.S.A. and the Netherlands both used computer voting when casting their votes. Many of the problems experienced by voters were noted, so to improve the implementation of the Irish system.

3 Progress to Date

The figure below represents the relationship between the main class, ElectionAlgorithm, and the rest of the classes. The Ballot and Candidate classes are also important to the architecture of the system as the process vital information about the election candidates and each voter's ballot. The remaining four classes provide the details that link both the Candidate and Ballot classes to ElectionAlgorithm.

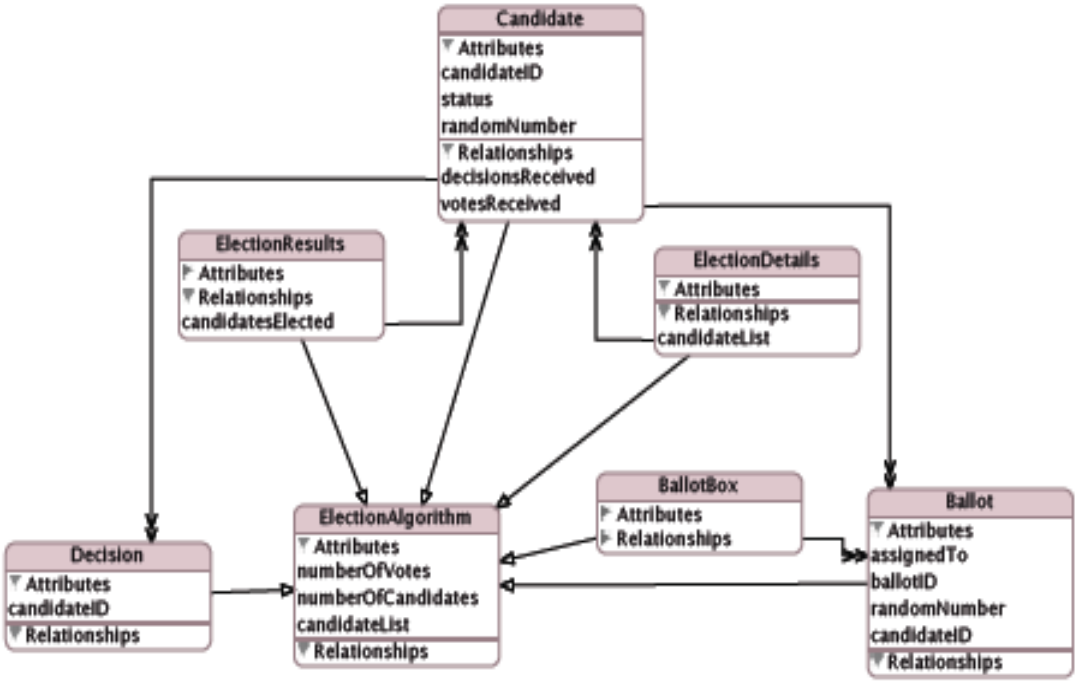

Figure 3: Class Diagram for the Irish Vote Counting Sub-System[1]

In the beginning, the main aim was to understand the Irish Voting System and the architecture of the system. This was an important aspect of the project as every eventuality must be catered for when implementing this system. It is these small cases that could make the difference in an election. Another aspect was to learn how to use the JML and ESC\Java2 tools because of their importance to the implementation of the Irish Voting System.

Figure 4 below shows the progress made in programming each method. As the smaller classes were programmed first this assisted the learning of the JML and ESC\Java2 tools. Also, the organisation of unit tests of all the methods and variables in these classes using JUnit has begun. The testing of these methods ensures that all the methods work for the small cases that were found above.

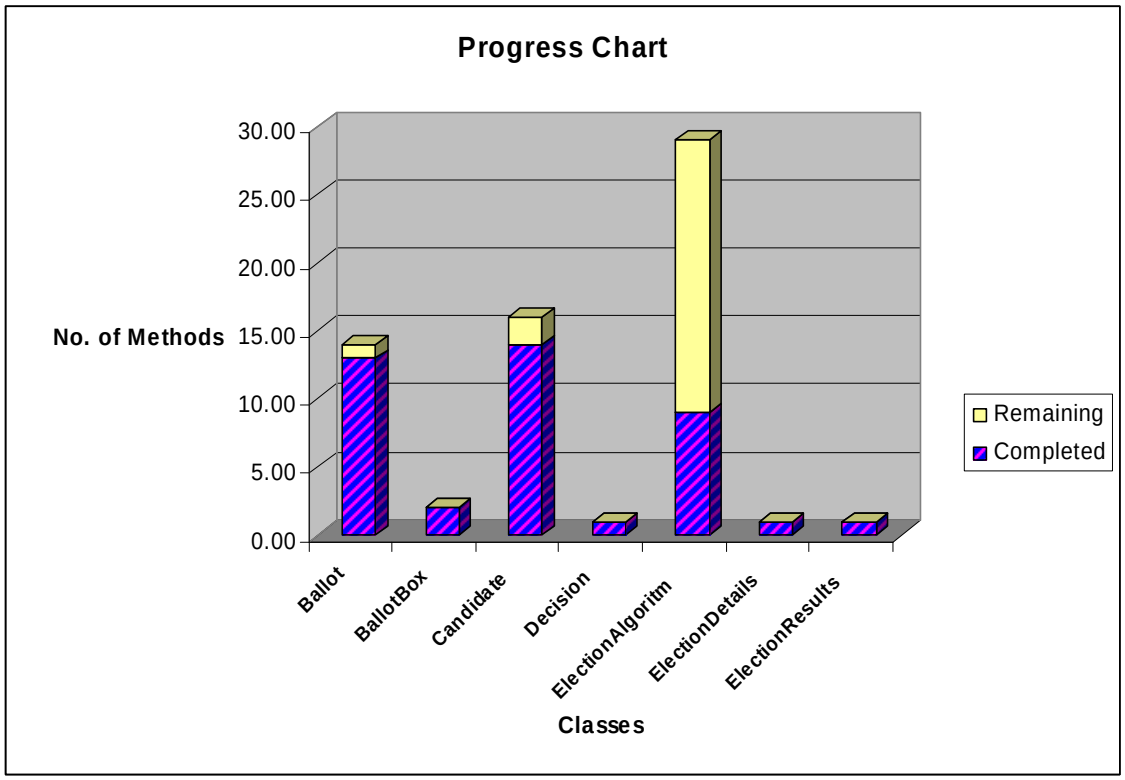

Figure 4: Progress of each class

3.1 Specification Faults

A common place in programming is finding faults in the project specification. One of the faults found in Dermot Cochran's specifications are discussed below. All specification faults can be found in Appendix A.

load( ): `//@ requires 0 < listSize;`

> The precondition *0 <= listSize* from was refined to *0 < listSize* because there must be at least one preference in the list at all times. The list lengths of both the *candidateIDList* and *preferenceList* arrays are set as *listSize*. The array *candidateIDList* must be non empty, and is therefore is non-null. A precondition is added to ensure *uniqueID* is the same as *ballotID*. This must be done as load must make the ballot valid as a *ballotID* of 0 is invalid.

3.2 Design

As specifications faults occur, design changes may have to be made, to ensure validation of the system. One of the design changes that have been made due to the specifications faults found in Appendix A, is discussed below. All design changes are noted in Appendix B.

Ballot( ): `//@ ensures ballotID == 0;`

It was decided that a ballot with an ID of 0 was an invalid ballot. This decision meant that the constructors post condition was then verified. A consequence of this change meant that the invariant of *ballotID* was modified to suit the above post condition. Again, *numberOfPreferences == 0* means an invalid *numberOfPreferences* and thus an invalid/uninitialised ballot.

3.3 Verification

ESC\Java2 does not provide operations to check all of the JML operators. A unit test must be performed instead to validate methods that contain any of these JML operators. Methods that contain any these operators in Dermot Cochran's specifications are discussed below.

Candidate Class

1 *getTotalVote( ):*

The *\sum* operator is a quantifier that returns the sum of the values that are given, where the values satisfy a set range [8,10]. As ESC/Java2 cannot check assertions that contain the *\sum* operator, we will test this method instead for validation. A JML expression *//@ nowarn Post* is added to the specification of the *getTotalVote( )* method, so ESC\Java2 does not give a warning about the post condition.

4 Future Work

The aim of the project is to implement the Irish voting system according to the specifications of Dermot Cochran. The main class of the specification, ElectionAlgorithm, has to be programmed. This method is the nerve centre of the project, as most of the system relies on it. I will also finish the testing of the small classes, as well as ElectionAlgorithm. To ensure that the system will work properly at all times, carefully selected inputs are used during unit testing. An EBON specification of the system would give readers an understanding of how the system works.

Of course the main objective of the project is to complete the mandatory goals, as specified in Section 2.1. If time permits, all of the discretionary goals may be completed. If the discretionary goals could be completed, the validation of system would benefit. It is unlikely that the exceptional goals will be attempted before the end.

Table 1. Gantt Chart detailing the progress of each task

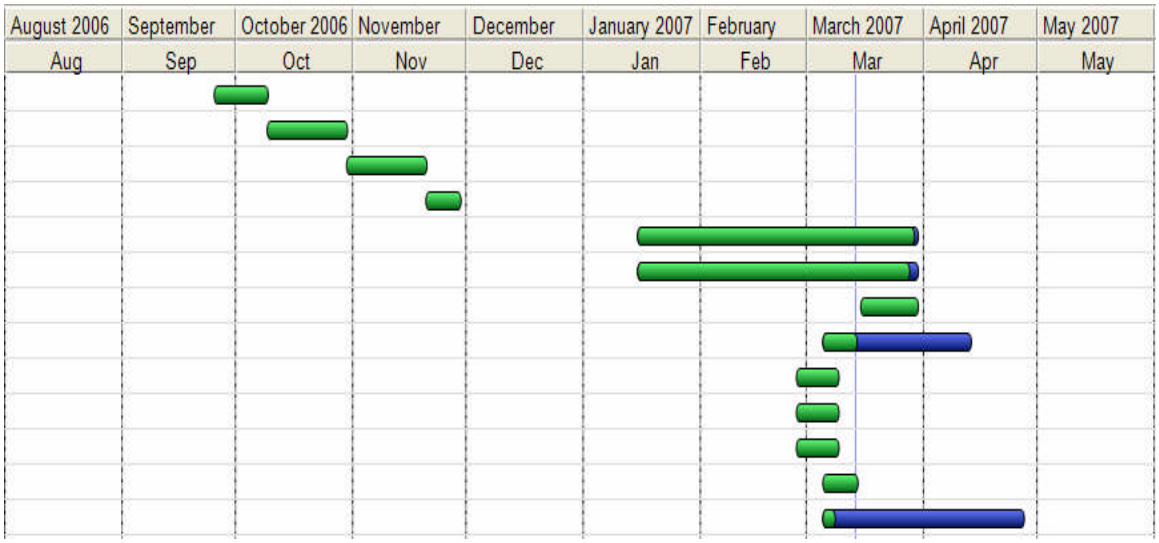

Table 2. Key to each row of the above table

#	Name	Start	Finish	Duration	Complete
1	Research the Irish Voting System	25/09/2006	09/10/2006	11.0 d	100.0%
2	Study the Specification	09/10/2006	30/10/2006	16.0 d	100.0%
3	Learn ESC/Java2 and JML tools	30/10/2006	20/11/2006	16.0 d	100.0%
4	Project Management Report	20/11/2006	29/11/2006	8.0 d	100.0%
5	Candidate Class	15/01/2007	30/03/2007	55.0 d	98.0%
6	Ballot Class	15/01/2007	30/03/2007	55.0 d	95.0%
7	BallotBox Class	15/03/2007	30/03/2007	12.0 d	100.0%
8	ElectionAlgorithm Class	05/03/2007	13/04/2007	30.0 d	25.0%
9	ElectionResults Class	26/02/2007	09/03/2007	10.0 d	100.0%
10	ElectionDetails Class	26/02/2007	09/03/2007	10.0 d	100.0%
11	Decision Class	26/02/2007	09/03/2007	10.0 d	100.0%
12	Interim Report	05/03/2007	14/03/2007	8.0 d	100.0%
13	Testing	05/03/2007	27/04/2007	40.0 d	10.0%

5 References

1. Dermot Cochran. "Secure Internet Voting in Ireland Using the Open Source Kiezen op Afstand (KOA) Remote Voting System" MSc. Dissertation Report, University College Dublin, 2006. http://secure.ucd.ie/documents/proposals/reports/Cochran06.pdf

2. Joe Kiniry, Alan Morkan, Dermot Cochran, Fintan Fairmichael, Patrice Chalin, Martijn Oostdijk, and Engelbert Hubbers. "The KOA Remote Voting System: A Summary of Work To Date" Proceedings of Trustworthy Global Computing (TGC 2006). Lucca, Italy, 2006.

3. Commission of Electronic Voting in Ireland: http://www.cev.ie/

4. Electoral (Amendment) Act 2004: http://www.cev.ie/htm/about/04act.pdf

5. Irelands Department of Environment and Local Government, http://www.environ.ie/elections/dailelect.html#sys

6. The Java Modeling Language (JML) home page, http://www.jmlspecs.org

7. The ESC/Java2 home page, http://secure.ucd.ie/products/opensource/ESCJava2/

8. The JUnit home page, http://www.junit.org/

9. Cok, D., Poll, E. & Kiniry, J. (2004), Introduction to JML , http://secure.ucd.ie/documents/tutorials/slides/1_intro_jml.pdf

10. Cok, D., Poll, E. & Kiniry, J. (2004), ESC/Java2 Uses and Features, http://secure.ucd.ie/products/opensource/ESCJava2/ESCTools/slides/2_tool.pdf

11. Burdy, L., Cheon, Y., Cok, D.R., Ernst, M.D., Kiniry, J.R., Leavens, G.T., Leino, K.R.M. & Poll, E. (2005), "An Overview of JML Tools and Applications", ftp://ftp.cs.iastate.edu/pub/leavens/JML/sttt04.pdf

12. Cheon, Y. & Leavens, G.T. (2006), Design by Contract with JML, ftp://ftp.cs.iastate.edu/pub/leavens/JML/jmldbc.pdf

13. Design By Contract: http://archive.eiffel.com/doc/manuals/technology/contract/

14. Kim Waldén and Jean-Marc Nerson. "Seamless Object-Oriented Software Architecture---Analysis and Design of Reliable Systems" Prentice-Hall, Inc., 1995. The BON specification language website: http://www.bon-method.com/

15. The Extended BON project: http://ebon.sourceforge.net/

16. The Verified Software Repository Project: http://www.fmnet.info/vsr-net/

17. The KindSoftware Coding Standard: http://secure.ucd.ie/documents/whitepapers/code_standards/

6 Appendices

A Specification Faults

Faults that have occurred in the project JML specifications.

A1 Ballot.java

2 *preferenceList*: `/*@ spec_public non_null@*/ long[] preferenceList = new long [0];`

This array can not be empty, hence non-null is used. Since the length of *preferenceList* must be equal to *numberOfPreferences*, the array is initialised to 0.

3 *numberOfPreferences*: `//@ public invariant (0 == numberOfPreferences) | (0 < numberOfPreferences);`

This variable should probably be an integer instead of a long because an array length is an integer and this field relates to *preferenceList's* length. Also, *numberOfPreferences == 0* means an invalid *numberOfPreferences* and thus an invalid/uninitialised ballot.

4 *candidateIDAtCount:* `//@ public invariant candidateIDAtCount.length == MAXIMUM_ROUNDS_OF_COUNTING;`

The length of this array is set using the variable *MAXIMUM_ROUNDS_OF_COUNTING* as there are only so many rounds of counting possible.

5 *ballotID:* `//@ public invariant (ballotID == 0) | (0 < ballotID);`

A ballot ID can either be 0 or the number assigned to it. An ID of 0 encodes an uninitialised ballot.

6 *transfer( ):* `//@ requires countNumber < MAXIMUM_ROUNDS_OF_COUNTING;`

An upper bound precondition is added to the variable *countNumber*.

7 *load( ):* `//@ requires 0 < listSize;`

The precondition *0 <= listSize* from was refined to *0 < listSize* because there must be at least one preference in the list at all times. The list lengths of both the *candidateIDList* and *preferenceList* arrays are set as *listSize*. The array *candidateIDList* must be non empty, and is therefore is non-null. A precondition is added to ensure *uniqueID* is the same as *ballotID*. This must be done as load must make the ballot valid as a *ballotID* of 0 is invalid.

A2 Candidate.java

8 *candidateID:*
```
//@  public constraint ((state != UNASSIGNED) &&
(\old(state) == state)) ==>
```

The constraint *old(state) == state* was added, as it allows the variable to be initialised.

9 *lastSetAddedCountNumber:*
```
//@ public constraint
\old(lastSetAddedCountNumber) <= lastSetAddedCountNumber;
```

There seems to be a bug in this constraint assertion, as it must hold for all methods (pure or otherwise), and the JML Reference Manual indicates that constraints generally are reflexive and transitive, and '<' is not reflexive.

10 *votesAdded( ) & votesRemoved( ):* `//@ public invariant votesAdded.length == MAXCOUNT; //@ public invariant votesRemoved.length == MAXCOUNT;`

The invariant *votesAdded.length == MAXCOUNT* was added as an upper bound to these arrays. A problem occurred when *votesRemoved* pointed to the same integer array as *votesAdded* in memory. As this should not happen, invariants were added . Another way to help solve this problem for each incident of *Candidate* was that each *Candidate* is assigned as the owner of both of the integer arrays, *votesAdded( ) & votesRemoved( ):* . This means that only the owner of these arrays may view or access the information stored in them.

11 *addVote( ) & removeVote( ):* `//@ requires 0 <= numberOfVotes;`

A precondition was added to set a lower bound on the variable *numberOfVotes* to conform to invariant `//@ public invariant (\forall int i; 0 < i && i < MAXCOUNT; 0 <= votesAdded[i]);`.

B Design Changes

Design changes that were included in the project JML specifications.

B1 Ballot.java

12 *MAXIMUM_ROUNDS_OF_COUNTING:* **`public static final int`** *`MAXIMUM_ROUNDS_OF_COUNTING;`*

This variable was added as there are only so many rounds of counting possible. As a result an upper bound is created on the array *candidateIDAtCount*.

13 *Ballot( ):* `//@ ensures ballotID == 0;`

It was decided that a ballot with an ID of 0 was an invalid ballot. This decision meant that the constructors post condition was then verified. A consequence of this change meant that the invariant of *ballotID* was modified to suit the above post condition. Again, *numberOfPreferences == 0* means an invalid *numberOfPreferences* and thus an invalid/uninitialised ballot.

shiftPreferenceList( ): `//@ assignable preferenceList[*];`

This method is added as a helper method to the method *transfer( )* to conform to the invariant *candidateID == preferenceList[positionInList]*. A helper method is allowed to adjust the values of variables even though they may be violating invariants.

B2 Candidate.java

15 *getUniqueNumber( ):* `//@ ensures \result >= 0;`

This helper method was added to assign a number to the variable *randomNumber*. The value for each candidate is just incremented, so this value is not random, but it does conform to the invariant *`(a.randomNumber == b.randomNumber) <==> (a == b).`*

16 *votesAdded( ) & votesRemoved( ):* `//@ public invariant votesRemoved.owner == votesAdded.owner;`

To prevent *Candidate* objects accessing the information stored by different *Candidate* objects, each *Candidate* object is assigned as the owner of both of the integer arrays *votesAdded( ) & votesRemoved( )*. This means that only the owner of these arrays may view or access the information stored in them.