Formal Techniques in a Remote Voting System

Joseph R. Kiniry, Alan E. Morkan and Dermot Cochran
School of Computer Science & Informatics
University College Dublin
Belfield, Dublin 4, Ireland

Martijn Oostdijk and Engelbert Hubbers
Institute of Information & Computing Sciences
Radboud University Nijmegen
Postbus 9010, 6500GL Nijmegen, The Netherlands

ABSTRACT
Kiezen op Afstand[1] (KOA) is a Free Software, remote voting system developed for the Dutch government in 2003/2004. In addition to being Open Source, key components have been, or are currently being formally specified and verified. These include a tally system and a modeling of the Irish electoral system. In this paper, we describe the formal techniques incorporated during the development of components of the KOA system. It also includes continuing work including the development of a platform for trustworthy voting from a mobile phone.

1. INTRODUCTION
KOA was used in the European Parliamentary election of June 2004. It was limited to expatriates and was subsequently released under the GNU General Public Licence.

A tally system for KOA was independently developed which was formally specified and verified. Since the release of the system under the GPL, the main KOA system has been re-completed[2] and specified using JML [1]. A component for use in Irish elections has also been formally specified. These verification efforts and the continuing work on the KOA system will be described in the subsequent sections.

2. TALLY SYSTEM VERIFICATION
The Dutch government decided that a separate vote counting subsystem should be implemented in isolation by a third party. This tally application would allow the vote counting to be independently verified. This was built and formally verified using JML and the ESC/Java2 [4] tool. The application consists of some 30 classes grouped into three categories: data structures, user interface, and tasks.

The data structure classes are easily described using JML. Typical concepts from the domain of voting, such as candidate, district and municipality can be modeled with detailed JML specifications. An example invariant in Candidate.java is:

```
/*@ invariant my_gender == MALE
  @     || my_gender == FEMALE
  @     || my_gender == UNKNOWN; */
```

The different tasks associated with counting votes were implemented by individual classes. After successful completion of a task, the application state is changed. A task can only be started if the application is in an appropriate state. The life-cycle model of the application that therefore emerges is maintained in the main class of the application inside a simple integral field. This life-cycle model can be specified in JML using invariants and constraints. For instance, such an invariant could read: ‘after the application reaches the “keys imported state”, the private key field is no longer null’. This is stated in MenuPanel.java as follows:

```
/*@ invariant
  @ (state >= PRIVATE_KEY_IMPORTED_STATE
  @  ==> privateKey != null); */
```

Finally, a graphical user interface is usually not very amenable to formal specification. Nonetheless, some light-weight specifications were written.

When the KOA vote counting system was being designed, precedence was given to verifying the core units. These were designed by contract and as a result have good specification coverage. The remaining parts, however, were only lightly annotated with JML notation. Table 1 summarizes the size (in number of classes and methods), complexity (non-comment size of source (NCSS)), and specification coverage of the three subsystems, as measured with the JavaNCSS tool.

Due to the time constraints, verification was only attempted with the core modules. Verification coverage of the core sub-

[1] “Kiezen op Afstand” is literally translated from Dutch as “Remote Voting.”

[2] Roughly 10% of the original code was proprietary and owned by LogicaCMG, the company who developed the main KOA system on behalf of the Dutch government. Therefore, it was necessary to reverse engineer the missing functionality.

	File I/O	Graphical I/O	Core
Classes	8	13	6
Methods	154	200	83
NCSS	837	1599	395
Specs	446	172	529
Specs:NCSS	1:2	1:10	5:4

Table 1: KOA initial release system summary

system was good, but not 100%. Approximately 10% of the core methods were unverified due to issues with ESC/Java2's Simplify theorem prover (i.e., either the prover did not terminate or terminated abnormally). Another 31% of the core methods had postconditions that could not be verified, typically due to completeness issues in ESC/Java2, and 12% of the methods failed to verify due to invariant issues. The remaining 47% of the core verified completely. Since 100% verification coverage was not possible in the timeframe of the original project, to ensure the KOA application was of the highest quality level possible, a large number unit tests were generated[3] for all core classes with the jmlunit [2] tool. A total of nearly 8,000 unit tests were generated, focusing on key values of the various datatypes (i.e., Candidate, District, etc.) and their dependent base types. These tests cover 100% of the core code and are 100% successful.

3. SPECIFICATION OF PR-STV

Naturally, there are relatively considerable variations in electoral systems between countries. This is the case between the Netherlands and Ireland. The Dutch Voting system is list based while Ireland uses Proportional Representation with a Single Transferable Vote (PR-STV).

Votáil is the Irish word for voting. The Votáil specification is a JML specification for the Irish vote counting system [3]. This formal specification is derived from the complete functional specification for the election count algorithm.

Thirty nine formal assertions were identified in the Commentary on Count Rules published by the Irish Department of Environment and Local Government. Each assertion expressed in JML was identified by a Javadoc comment. In addition, a state machine was specified so as to link all of the assertions together. Java classes were specified for the vote counting algorithm, to represent the ballot papers and candidates. This was then typechecked and checked for soundness using ESC/Java2.

4. MOBILE VOTING SPECIFICATION

The EU MOBIUS Project[4] focuses on several topics including the specification and verification of security properties at several levels.

As part of this work, the security properties, including a functional specification, for a MIDP-based remote voting application are in the process of being defined. An example of such a security property is: "The application must not have access to personal information (e.g., phonebook) on the mobile phone".

Additionally, a MIDP-based remote voting applet has been developed at UCD. This application has been reviewed and will be refactored, including the security and functional requirements expressed in JML, for incorporation into KOA.

5. CONCLUSION

We have presented a brief description of the formal techniques incorporated in the development of important components of the KOA remote voting system. While integrating the Votáil subsystem into the KOA system, and prior to the new full FLOSS foundation release of KOA, a number of new pieces of English documentation and functional specification must be written. Given that remote voting is a key case study in verified computing, we hope that the availability of such documentation and specification will provide additional motivation for researchers and developers to seriously consider using the KOA system as a foundation for Verified Verifiable Voting (VVV).

6. ACKNOWLEDGEMENTS

This work is being supported by the European Project Mobius within the frame of IST 6th Framework, national grants from the Science Foundation Ireland and Enterprise Ireland and by the Irish Research Council for Science, Engineering and Technology. This paper reflects only the authors' views and the Community is not liable for any use that may be made of the information contained therein.

7. REFERENCES

[1] Lilian Burdy, Yoonsik Cheon, David Cok, Michael Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll. An Overview of JML Tools and Applications. *International Journal on Software Tools for Technology Transfer*, Feb 2005.

[2] Yoonsik Cheon and Gary T. Leavens. A Simple and Practical Approach to Unit Testing: The JML and JUnit Way. In Boris Magnusson, editor, *Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP 2002)*, volume 2374 of *LNCS*, pages 231–255. pub-sv, Jun 2002.

[3] Dermot Cochran. Secure Internet Voting in Ireland using the Open Source Kiezen op Afstand (KOA) Remote Voting System. Master's thesis, University College Dublin, March 2006.

[4] Joseph R. Kiniry and David R. Cok. ESC/Java2: Uniting ESC/Java and JML: Progress and issues in building and using ESC/Java2 and a report on a case study involving the use of ESC/Java2 to verify portions of an Internet voting tally system. In *Construction and Analysis of Safe, Secure and Interoperable Smart Devices: International Workshop, CASSIS 2004*, volume 3362 of *lncs*. pub-sv, Jan 2005.

[3] The tool generates unit tests that deal with *interesting* values. Interesting values are generally boundary values for a given data type. For example, -1, 0, 1, *n* and *n+1* for an array of integers.

[4] The MOBIUS Project - http://mobius.inria.fr/