Formally Counting Votes (But Still Only Trusting Paper)

ICECCS'07 Auckland, NZ

Joe Kiniry
KindSoftware Research Group
Systems Research Group
CASL: Complex and Adaptive Systems Laboratory
School of Computer Science and Informatics
University College Dublin, Ireland

This work was funded in part by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS Project. This presentation reflects only the author's views; the Community is not liable for any use that may be made of the information contained therein.

Acknowledgments

- Engelbert Hubbers, Bart Jacobs, Martijn Oostdijk, Wolter Pieters (RUN); Patrice Chalin (Concordia)
- UCD postgraduate students Dermot Cochran, Fintan Fairmichael and Alan Morkan
- UCD undergraduate students Barry Denby, Conor Gallagher, and Patrick Tierney

Formal Methods and Computerized Voting

- While we have experience in the application of formal methods in computer-based voting systems in the Netherlands and Ireland...
- We are *computer scientist activists* that have been fighting against the introduction of corporation-driven, untrustworthy, computer-based voting in NL, IE, USA, and elsewhere.
- ***Computer-based voting must not be used for any important elections until we better understand its challenges and solutions.***

Our Work in “Solving” the Voting Problem

- Remote voting over a network incorporates many of the core challenges of trusted and trustworthy global computing.
  - Part of my reason for giving talks about this work is to recruit scientists to this very important work and help convince *scientists* decide to become *activist scientists*.
- Our contribution/foundation: Kiezen op Afstand (KOA)—a remote voting system developed for the Dutch government in 2003.

Formal Specification and Verification

- In addition to being Open Source, KOA is also (partially) formally specified and verified.
- The Dutch vote counting system was formally specified using JML and its correctness checked using ESC/Java2 and unit testing.
- The Irish vote counting system has since been specified using JML (by MSc student Dermot Cochran), and has been partially implemented and verified by a final year student (Patrick Tierney).

A Little History: e-Voting in NL

- NEDAP machines have been used in NL for over a decade for kiosk-based voting.
- The Dutch European Parliament elections in June 2004 permitted remote voting via the internet and telephone for expatriates.
  - The prior remote voting system was based upon postal ballots.
- KOA was designed, developed, tested, deployed, and managed by LogicaCMG under contract with the Dutch government.

Open Source Release of KOA

- The SoS Group at RUN was involved in a covert and overt security analysis of KOA.
- Recommendations were made in two reports written for the minister & parliament.
- In July 2004, the Dutch Government released the majority of the source code for the KOA system under the GNU General Public License (GPLv2), making it the first government-sponsored, fully implemented, Open Source internet voting system in the world.

Formal Specifications for KOA

- The tally application for the Dutch system consists of 30 classes, grouped into three categories:
  - data structures,
  - user interface, and
  - tasks.

Specifying Data Structures

- The data structure classes represented an excellent opportunity to write JML specifications and perform verification.
- Typical concepts from the domain of voting such as candidate, district and municipality are modeled with detailed JML specifications.
- Simple models like arrays are used as well.

Specifying Tasks

- The different tasks associated with counting votes are mapped to individual Java classes.
  - e.g., initialization, clear votes, import candidates, read public/private keypair, decrypt votes, count votes, write report
- 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.

Life Cycle Model

- The algorithm is specified in JML using an ASM, represented by a set of class and object invariants and constraints.
- The specification states that, on successful completion, the tally application went from "initial state" to "votes counted state".
- Thus, the theorem encoded by the tally application is the conjunction of invariants in the final "report generated" state:
  - all legal votes in the encrypted ballots have been successfully counted and reported

Irish Vote Counting System

- the Dutch Voting system is a list based voting system
  - voters vote for parties, not individuals
- Ireland uses Proportional Representation with a Single Transferable Vote (PR-STV)
  - *voters rank individuals by preference*
- the Scottish system is very similar to Irish one
  - recently developed in Nijmegen by SoS Group

Irish Vote Counting Specification

- 39 formal assertions were identified in the Count Rules published by the Irish Government.
- Each assertion was expressed in JML and identified and cross-referenced by a Javadoc comment.
- A state machine was specified so as to link the assertions together.

Specification of Vote Transfer Method

```
/**
 * Transfer votes from one candidate to another
 * @param fromCandidate Elected or excluded candidate
 * @param toCandidate Continuing candidate
 * @param numberOfVotes Number of votes to be transfered
 */

/*@ requires fromCandidate.getStatus() != Candidate.CONTINUING;
  @ requires toCandidate.getStatus() == Candidate.CONTINUING;
  @ requires numberOfVotes == getActualTransfers (fromCandidate, toCandidate) +
  @    getRoundedFractionalVote (fromCandidate, toCandidate)
  @ ensures fromCandidate.getTotalVote() ==
  @    \old (fromCandidate.getTotalVote()) - numberOfVotes;
  @ ensures toCandidate.getTotalVote() =
  @    \old (toCandidate.getTotalVote()) + numberOfVotes;
  @*/

protected void transferVotes(/*@ non_null @*/ Candidate fromCandidate,
                             /*@ non_null @*/ Candidate toCandidate,
                             long numberOfVotes);
```

How Many Votes to Transfer?

```
/**
 * Determine actual number of votes to transfer to this candidate, excluding
 * rounding up of fractional transfers
 *
 * @see requirement 21, section 7, item 3.1, page 24
 * @see requirement 22, section 7, item 3.2, page 25
 *
 * @design The votes in a surplus are transfered in proportion to
 *   the number of transfers available throughout the candidates ballot stack.
 *   The calculations are made using integer values because there is no concept
 *   of fractional votes or fractional transfer of votes, in the existing manual
 *   counting system.  If not all transferable votes are accounted for the
 *   highest remainders for each continuing candidate need to be examined
 *
 * @param fromCandidate Candidate from which to count the transfer
 * @param toCandidate Continuing candidate eligible to receive votes
 *
 * @return Number of votes to be transfered, excluding fractional transfer
 */
```

Votes Transfer Method

```
/*@ requires (state == COUNTING);
  @ requires (fromCandidate.getStatus() == Candidate.ELECTED) |
  @   (fromCandidate.getStatus() == Candidate.ELIMINATED)
  @ requires toCandidate.getStatus() == Candidate.CONTINUING;
  @ ensures ((fromCandidate.getStatus() == Candidate.ELECTED) &&
  @   (getSurplus(fromCandidate) < getTotalTransferableVotes(fromCandidate))) ==>
  @   (\result == (getSurplus (fromCandidate) *
  @      getPotentialTransfers (fromCandidate, toCandidate.getCandidateID()) /
  @      getTotalTransferableVotes (fromCandidate)));
  @ ensures ((fromCandidate.getStatus() == Candidate.ELIMINATED) ||
  @   (getTotalTransferableVotes(fromCandidate) <= getSurplus (fromCandidate)))  ==>
  @   (\result == (\num_of int j; 0 <= j && j < totalVotes;
  @     ballotsToCount[j].isAssignedTo(fromCandidate.getCandidateID()) &&
  @     getNextContinuingPreference(ballotsToCount[j]) == toCandidate.getCandidateID()));
  @*/
protected /*@ pure @*/ int getActualTransfers(/*@ non_null @*/ Candidate fromCandidate,
                                              /*@ non_null @*/ Candidate toCandidate);
```

Finding the Next Preference Candidate

```
/**
 * Gets the next preference continuing candidate.
 *
 * @design This is the _nearest_ next preference i.e.
 * filter the list of preferences to contain continuing candidates and then
 * get the next preference to a continuing candidate, if any.
 *
 * @param ballot Ballot paper from which to get the next preference
 *
 * @return Candidate ID of next continuing candidate or NONTRANSFERABLE
 */
/*@ requires state == COUNTING;
  @ ensures (\result == Ballot.NONTRANSFERABLE) <=!=>
  @   (\exists int k; 1 <= k && k <= ballot.remainingPreferences();
  @     (\result == ballot.getNextPreference(k)) &&
  @     (\forall int i; 1 <= i && i < k;
  @       isContinuingCandidateID(ballot.getNextPreference(i)) == false));
  @*/
protected long getNextContinuingPreference(/*@ non_null @*/ Ballot ballot);
```

Voting as a Grand Challenge Pilot Project

- written in Java, fully Open Source (GPL)
- critical application domain
- large but well-decomposed
  - ~550 classes but only ~36,000 NCSS
- small set of interesting core theorems
  - only eligible voters vote, they only vote once, all valid votes are counted, etc.

Voting as a Grand Challenge Pilot Project

- demands modern techniques
  - conservative use of concurrency
  - non-interference
  - confidentiality and declassification
- depends upon a set of useful APIs
  - crypto, EJB, database, simple GUI, etc.
- system and tools to be contributed to the Verified Software Repository

Alternative Systems

- Dutch REIS system
  - implemented in JavaScript
- OpenVotingConsortium EVM system
  - implemented in Python
- eVACS from Australia
  - implemented in C
    - Unfortunately, none of these systems have any (even semi-)formal specifications.

Ongoing and Future Work

- The security properties, including a functional specification, for a MIDP-based remote voting application are in the process of being defined.
  - High-level requirements are defined, but have not yet been refined to low-level specifications.
- We are interested in collaborations to formally specify and verify other voting systems (e.g., Prêt à Voter, American, etc.)
- Hosting research visitors and work with IEEE.

Systems-level Challenges

- *Trustworthy voting is about much more than the software and hardware involved in the voting system and process.*
- The non-computational aspects of voting (people, parties, organizations, ballot design, polling place organization, etc.) are of paramount import to voting systems.
- How can we specify, validate, and verify these non-computational aspects?

Questions and Comments?

- next steps?
  - other potential collaborators?
  - open theoretical challenges?
  - tool limitations?
  - funding opportunities?
  - publications and public relations?