Engineering a Distributed e-Voting System Architecture: Meeting Critical Requirements

J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy

Département Logiciels-Réseaux (LOR),
Telecom & Management SudParis,
9 rue Charles Fourier, 91011
Évry cedex, France

Abstract. Voting is a critical component of any democratic process; and electronic voting systems should be developed following best practices for critical system development. E-voting has illustrated the importance of formal software engineering in the development of complex systems: poorly engineered and poorly documented voting systems have had serious negative consequences for all system stakeholders. It is clear that the formal verification of e-voting system models would help to address problems associated with certification against standards, and would improve the trustworthiness of the final systems. However, it is not yet clear how best to carry out such formal modelling and verification in order to leverage the compositional nature of the problem, and manage the complexity of the task.

The choice of modelling language - for expressing the high level design and architecture of an e-voting system - poses many problems due to the complex mix of requirements that such a system is required to meet. Different modelling languages are more-or-less suited to the verification of different critical requirements. Thus, we report on a mixed model approach: where we address 3 different types of critical requirements using 3 different modelling languages and development strategies. Firstly, we report on network quality-of-service issues that are analyzed through simulation models. Secondly, we report on functional correctness of a counting process that can be validated through algebraic techniques. Finally, we report on the use of formal refinement to reason about the correctness of design steps when adding detail to an architecture model. To conclude, we acknowledge the main problem that arises from such a mixed-model approach to architecture verification: how can we be sure that the different models are coherent when we integrate them in a final implementation?

1 Introduction

1.1 Overview

The work presented in this paper is part of an applied research project in which the objective is to develop a prototype for an innovative e-voting system for use

H. Giese (Ed.): ISARCS 2010, LNCS 6150, pp. 89–108, 2010.

in France[1]. The project is constrained by existing rules, regulations, laws and standards that the specific elections are supposed to meet, including European recommendations[25]. A main goal is that the prototype demonstrates that such a system can be manufactured at reasonable cost, and that it meets the needs of the electorate. A secondary objective is to demonstrate the application of formal methods — as in [5,6] — in the engineering of the software in the e-voting system, which we consider to be critical[24].

The software process that we followed was that of rapid-prototyping, as dictated by the limited time frame of the project. However, we applied formal modelling techniques where we felt they would add rigour to the development process without compromising the time limits. The development was intended as a learning process where we would use different formal techniques as the need arose. Thus, as well as developing a prototype e-voting system that would help us to build a final version in the future, we would also develop a better understanding of the role of the different formal methods that would help us to follow a more formal development process in future development.

The main innovation in the system is concerned with allowing the voter to choose to vote at any official voting location (and not to be restricted to a single voting station). The challenge with this innovation is to design a distributed architecture which is robust against denial of service attacks during the voting process. Currently, in France, in order to meet the requirement that no person can have more than one vote counted during an election, a person can vote at most one time. This is enforced by having a list of all people who have voted stored locally at each voting station. By allowing voters to *VoteAnywhere*, we chose not to enforce the restriction on voting only one time; as this would require either use of a network during voting in order to allow sharing of information between voting locations, or use of some complex protocol involving physical tokens that voters would "pay" in order to vote. Rather, we chose to allow revoting and to guarantee that only a single vote for each voter is counted after the voting process is terminated.*Revoting* is not strictly necessary to permit a voter to *VoteAnywhere*, but — as we demonstrate later in this paper — it simplifies the development of the system, as well as offering some advantages to the voter.

The solution that we propose does not completely remove the need for some sort of global functionality during the voting process: we require the use of clocks that are synchronized between election locations; but demonstrate that this solution is much more robust against a denial of service attack.

1.2 Structure of Paper

In section 2 we review the specific innovative features of our chosen system and comment on the main architectural concerns. In section 3 we provide a brief summary of previous research on distributed and remote e-voting system architectures. Section 4 focuses on the key requirement that the e-voting system

[1] The system documentation is in French and we have translated the main concepts and components into English. Where multiple translations are equally reasnaoble, we have noted this in the text.

should be — as far as possible — robust against denial-of-service attacks. In particular, section 4 shows that support for simulation is a major advantage when choosing a modelling language with operational semantics; in our study we used Estelle[17]. In section 5, we report on the formal specification of the fundamental data (and data transformations) that are used in the counting (tabulation) process. In particular, we illustrate how the simple algebraic specification of invariant properties can aid validation[12,7] and help developers avoid types of tabulation error that are common to e-voting systems. Section 6 illustrates the use of refinement (with Event-B and the RODIN toolset[1]) for the specification and verification of a design transformation step. Section 7 reviews the development of a prototype implementation where the main difficulty was a coherent integration of multiple views as specified by our different modelling languages. We conclude the paper in section 8.

2 Revote Anywhere (By Procuration): Our Specific Requirements and Architecture Concerns

We consider the requirements for secrecy and accuracy[8] to be fundamental to all voting systems of interest to the research community. In all discussions that follow with regards to voting systems, it is implicit that no additional requirement should compromise the need for secrecy and accuracy.

We also consider *quality-of-service* to be a critical property in any voting system - the effort required to vote must not discourage electors from engaging in the voting process. In particular, the time that is required to vote must not be *unreasonable*.

In the following, we introduce 2 voting innovations (for France) — permitting voters to vote at numerous different locations and at numerous different times — and illustrate some of the problems that may arise when these innovations have to be integrated with existing features, such as allowing a third party to vote on behalf of an elector.

2.1 VoteAnywhere: A First Innovation

Restricting each elector to vote at a single specific location can have a significant negative impact on voter turnout. Providing flexibility in where electors can go to vote should improve voter turnout, and this is the major high-level objective of the *VoteAnywhere* innovation.

This paper is not proposing remote electronic voting where electors are able to vote over the internet — in the next section we review many of the problems that can arise if such unconstrained remote voting is allowed. We agree with the conclusions in a review paper — *The Development of Remote E-Voting Around the World: A Review of Roads and Directions*[21] — that: "Overall remote electronic voting has not reached the maturity to be applied in large-scale elections of major importance." Rather, we are proposing that electors be allowed to vote at any authorised polling station. This *VoteAnywhere* requirement provides many

of the advantages of remote voting whilst not being vulnerable to most of the weaknesses[13,30]. We note that a less general variation on *VoteAnywhere* functionality is the use of *remote voting centers*[30], for "voters far from their home precincts". This approach does not meet our objective of allowing all electors to vote at any authorised voting center (but it does illustrate that the need for remote voting is well acknowledged.)

Two other requirements are key to the development of our prototype system: *Revote* and *Procuration*. In the subsections that follow we summarise the potential interactions between each of these features[14].

2.2 VoteAnywhere with ReVote

`Revote` facilitates the implementation of a system that meets our `VoteAnywhere` requirement without risk of denial of service attacks. However, it also provides additional benefits to the voter when we consider elections run over a long time period. Restricting electors to vote during a narrow time frame can reduce turnout. However, widening the times when electors can vote (as with early voting in the USA) introduces the problem that electors may be discouraged from voting early because they do not have an opportunity to change their vote at a later time (while the voting process is still open.) The main objective of the *ReVote* innovation is to encourage early voting (and consequently improve turnout) by permitting an elector to revote if they wish to change a previous recorded ballot. A fundamental requirement is that only a single vote is counted for each elector. In our chosen system we refine this fundamental requirement into a rule that states that if an elector votes multiple times then only the last vote recorded by this elector will be counted.

Provided that an elector has to vote at the same polling station then there should be no problem in identifying which vote was the last recorded when a *ReVote* occurs. Some obvious options are:

1. Use a local clock to stamp each signature.
2. Use a local counter to stamp each signature.
3. Use a "destructive-write" so that a signed bulletin[2] added to the local urn automatically results in the destruction of any bulletin that shares the same signature already in the urn[3].

However, integrating *VoteAnywhere* with *ReVote* poses problems in all three of the optional designs above:

1. Local clocks would need to be synchronised or replaced by a global clock.

2.&3. Require a reliable non-local network for communication of data between distributed polling booths.

In section 4 we show that option 1 is the only acceptable (and feasible) solution to meeting all our requirements.

[2] A bulletin is also know as a ballot.

[3] An urn is also known as a ballot box.

2.3 Procuration, ReVote and VoteAnywhere: A Feature Interaction

Procuration[4] is the feature that permits one elector (the elector-by-procuration[5]) to vote on behalf of another elector. In many elections, procuration does not necessarily prohibit an elector from voting. In France, for example, the elector may be able to go to a polling station and vote, provided that the elector-by-procuration has not already done so. Given a reliable non-local communication network then there are no undesirable interactions between *Procuration* and *VoteAnywhere* as a central voter list could guarantee that the elector and the elector-by-procuration cannot record two suffrages "at the same time" at different polling stations; in the same way that this is currently guaranteed by local voter lists at each polling station.

There is a clear undesirable interaction between *Procuration* and *ReVote* during the election process. In the first instance, an elector may be denied the right to record a suffrage whilst in the second instance an elector must never be denied the right to record a suffrage. Consequently, to provide the *ReVote* feature it may be necessary to change the existing regulations with respect to *Procuration*.

Using local clocks to implement *ReVote Anywhere* can lead to additional requirements when combined with *Procuration*. Without *procuration*, a design which uses global clocks to time-stamp ballots can safely make the assumption that a "single elector" cannot be in two places at once. As a consequence, the accuracy of the clocks is not critical and inexpensive solutions should be considered. However, with *Procuration* and *VoteAnywhere* it is possible that an elector is in two different polling stations at the same time[6]. This scenario may require much more accurate (and much more expensive) global clocks.

In our chosen system, we address these potential problems by adhering to the spirit of procuration - a vote from the original (non-procured) elector should take priority over a procured vote, irrespective of the time at which they are recorded.

2.4 Audits and Recounts

A main weakness of our proposed system is that the votes cannot be recounted by hand. The voter does have a paper record of their vote but it is impossible for them to be decrypted and hand counted — the encrypted votes must be counted as a whole before the result is decrypted.

The paper record of the vote allows a limited type of verifiability (or audit) — a voter can verify that their vote was counted after the election, but they cannot verify that it was correctly counted. The voter can also verify that the encryption process correctly records votes (during the voting process).

[4] Procuration is also known as proxy voting or vote delegation.
[5] The elector-by-procuration is also known as the proxy voter.
[6] This arises if the elector goes to one polling station and the elector-by-procuration goes to another.

3 Distributed/Remote E-Voting Systems: Architecture and Design Issues

In all distributed voting systems, denial-of-service of the underlying communication architecture is a major threat. In remote voting there are also increased threats of voter coercion and/or the voting machine being untrustworthy. In the *VoteAnywhere* system, because electors vote in a controlled polling station, voter coercion should be no greater an issue than with traditional voting. Trusting e-voting machines is a major concern for all voting systems, but one which is much more serious for remote voting where the machines are not under the direct control of the voting authorities.

The design of remote electonic voting machines — requiring a network for communication between machines — is clearly a much more complex problem than the design of standalone machines. In the remainder of this section we review some of the most relevant previous research in these areas.

3.1 Denial-of-Service

In 1998, Susan King Roth identified voter disenfranchisement as a main risk of poorly designed e-voting systems[28]. Her analysis raised interesting questions with respect to poorly designed machines discouraging voter participation. This is particularly relevant when we consider the requirement that voting takes a reasonable amount of time.

In 2000, Hoffman asked *Internet Voting: Will it Spur or Corrupt Democracy?*[16], and commented on the perceived risk of denial-of-service attacks: "Imagine what a concerted denial of service attack might do to an election with Internet/Web-based voting . . . ".

In 2003 the design of an internet voting system is proposed in *REVS — A Robust Electronic Voting System*[19]. The authors write that they have designed: "a robust electronic voting system . . . that tolerates failures in communications and servers while maintaining all desired properties of a voting system." However, the key issue of anonymity is mentioned only briefly in the conclusions, where the authors state that "REVS can beneficiate from a more sophisticated anonymity mechanism". In 2004, further analysis of the REVS architecture identified weaknesses inherent in the design due to voter information being centralised[33], which introduces additional dependency on the underlying communication network.

In the same year, Chen et al. proposed the *Design of a secure anonymous Internet voting system*[9] and claim that their "scheme does not require a special voting channel and communications can occur entirely over the current Internet". They do consider the robustness of their system with respect to election disruption (through voter behaviour): "Even if a voter intends to disrupt the election, there is no way to do it. The only way to disrupt the elections is for the voter to keep sending ballots to the TC and SC." They then continue by explicitly forbidding re-voting: " However, the TC and SC will verify the validity of the voter-pseudonym signature and will not allow the same voter-pseudonym to

vote twice." This approach — which ignores potential denial of service attacks on the network (independent of voter behaviour) and which forbids revoting — is very different to our proposal.

In *Verifiable Anonymous Vote Submission*[36] the *REVS* architecture is adapted to better deal with anonymity and verifiability. This work is based on two previous anonymization architectures — Mix Nets and Mix Rings — which were not originally intended for e-voting systems but which now form the central design feature of many proposals for remote electronic voting. In general, the design of such systems focuses on security aspects rather than on denial-of-service issues.

We note that relying on the internet provides opportunities for attack from foreign agents. Jefferson et al. write in *Analyzing Internet Security*[18]: "Because the internet is independent of national boundaries, an election held over the internet is vulnerable to attacks from anywhere in the world."

In 2004, Selker and Goler report on *The SAVE system — secure architecture for voting electronically*[31]: " This voting architecture provides a means to vote over open networks in a way that is reliable, secure, and private. " Their proposal is based on demonstrating that — through n-version redundancy techniques — there is no single point of failure in their system. However, their proposed architecture is not robust against denial-of-service attacks.

Two years later, in 2006, another article — *E-voting in Estonia 2005. The first Practice of Country-wide binding Internet Voting in the World*[23] — reports on the co-ordination activities that are necessary when relying on the internet during e-voting: "System and network monitoring was performed by different parties on different levels during the e-voting period on a 24h basis. All major e-service providers (e.g. banks) and Internet operators were involved in the process with monitoring the overall "health" of Internet network traffic loads, analysis of possible Trojans/viruses etc." They do not detail the contingency plans if their network fails during an election; but it is likely that the election would have to be aborted and re-run. Thus, one could say that their design is not dependable. The notion of "Design for dependability" appears in an article by Bryans et al. in 2006[4], where they consider the importance of robustness and fault-tolerance. They conclude that: "... aborted elections are still failures."

Qadah and Taha propose an alternative remote e-voting architecture and illustrate how mobile devices can be used as voting client machines[27]. However, they do note that their implementation — using public wireless networks — is not suitable for secure elections: "... for highly secure elections, such as political ones, voters need to access the e-voting system through secure channels including the use of secure client devices located at secure polling locations and connected to the e-voting system through secure Intranets/private networks". It is interesting to note that they focus on the security of channels and networks without explicitly mentioning reliability.

3.2 Coercion and Anonymity

Coercion is a major issue in any voting system where voters are able to demonstrate how they have voted. In most traditional systems, specific procedures have

evolved in order to minimize the risk of coercion. Anonymous voting is the most widely applied technique for mitigating coercion — if all ballots are anonymous then there is no way for an elector to demonstrate (to a coercer) how they have voted. Thus, even if an elector is coerced there is no risk that the coercer can verify if the coercion has worked.

Remote e-voting would appear to increase the risk of coercion. Maaten, in *Towards Remote E-Voting: Estonian case*[22] provides evidence of coercion in remote e-voting: "During the last elections in Estonia some vote-buying incidents became public."

The design of a secure (coercion-free) remote e-voting system is proposed in *Civitas: A Secure Remote Voting System*[10]. The paper addresses one of the major problems with remote voting: how can one ensure that voters cannot be coerced when the voting location is unsupervised? In particular they use the requirement that "voters cannot prove whether or how they voted, even if they can interact with the adversary while voting." It should be noted that the architecture may be susceptible to denial-of-service attacks: "Civitas does not guarantee availability of either election authorities or the results of an election.

Our proposed system introduces no significant risks — over the paper system — with respect to anonymous voting. However, there is a coercion attack which could be used to force a voter to make a random vote: as a voter has a printed record of their vote against a random permutation of candidates it is possible that they would be obliged to vote randomly if an attacker forces them to record a particular sequence of preferences. This attack could not force a voter to record a particular vote because the attacker has no way of knowing how the preferences have been permutated but it does introduce an additional risk.

3.3 Other Related Issues

In 2002, Rubin analyses the *Security Considerations for Remote Electronic Voting over the Internet*[29] and concludes that: " . . . the technology does not yet exist to enable remote electronic voting in public elections." We argue that, 8 years later, there have been no major technological advances that would require one to change this conclusion.

In *Swiss E-Voting Pilot Projects: Evaluation, Situation Analysis and How to Proceed*[3], the authors note that: "Parliament demanded of e-voting a similar level of security to that of postal voting." As postal voting is the most problematic with respect to meeting requirements, it should not be a surprise that they conclude: " The required benchmark was exceeded in the pilot trials." We argue that the benchmark for comparison must be set to a level equivalent to the best paper systems.

In *e-Voting Requirements and Implementation*[2], "the complexity of the deployment of e-voting systems and the inherent security issues that arise from the underlying distributed system" is considered. An architecture that focuses on "the security of the election servers and the channels between client machines and the servers" is proposed. Unfortunately, the authors identify a major weakness in their

architecture (and with remote voting, in general) — they cannot guarantee the security of the client machine from which a vote is cast.

4 Denial-of-Service: Our Specific Requirements

4.1 Our Specific Requirements

Elections that depend on distributed communicating (sub)systems are open to denial-of-service attacks on the underlying communication architecture. The consequences of such attacks are likely to be critical during the voting process — if electors are unable to vote for long periods of time then the election will almost certainly have to be re-run. Contrastingly, such attacks occuring before or after voting should not, if properly managed, have such serious consequences.

We propose that distributed voting systems must not depend on a reliable internet connection during the voting process in order to meet functional and non-functional requirements. In particular, no part of the voting process should depend on the sending or receiving of information on the internet (during the vote). This is the only way to guarantee that successful denial-of-service attacks cannot prevent electors from voting in a reasonable amount of time.

4.2 Simulation of Estelle Architecture Models

In previously reported research [13,14], through simulations of the formal models (written in Estelle[17]), we established that certain architectures could not provide an acceptable quality of service (to the voter) when the underlying communication network was open to denial-of-service attacks during voting. However, one particular architecture — using clocks for timestamping, but no other network communications during voting — appeared (through analysis of the simulation data) to be a possible solution to our problem.

In figure 1 we show the three alternative architectures that we modelled, in Estelle, for simulation. The first uses global lists for recording which electors (who are entitled to vote) have already voted and for the choice of candidates (vote options) offered to them. The second uses local lists to record information of/for electors who have gone to their local voting station; whilst using global lists for those who have chosen to vote elsewhere. The third has local copies of all electoral and candidate lists. The diagrams are generated from the semantics of the formal Estelle specifications. The key difference between the architectures is concerned with when the network is un use. Architecture one requires a network connection for every elector. Architecture two requires a network connection for electors who have chosen to vote at their non-default polling station. Architecture three never depends on a network connection during the voting process.

It is important to note that our simulation models — including the architecture of our chosen system — abstracted away from key aspects of e-voting systems that require the use of security protocols. As we evolved our design to incorporate these aspects, it was critical that we ensured the soundness of the abstraction: no new behaviour should introduce a need for communication across an unreliable network during the vote process.

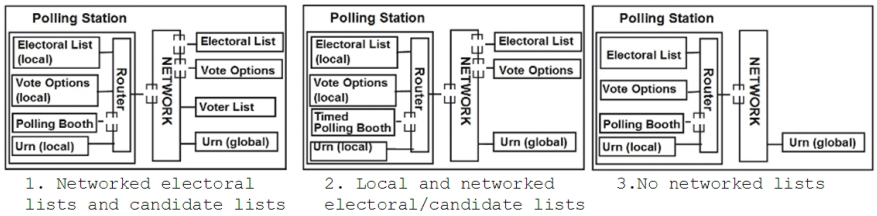

Fig. 1. Abstract Architecture Alternatives

4.3 Final Design: When Do We Need a Network?

Our design went through a number of stages. In figure 2 we show that consistency with our abstract architecture — with respect to network dependency — was maintained as further requirements for security, authentication, encryption and voter verifiability were added to the system. The key is that communication between these additional components (and the polling stations) is not necessary during the voting process. All new components are connected to local urns in each of the polling stations;

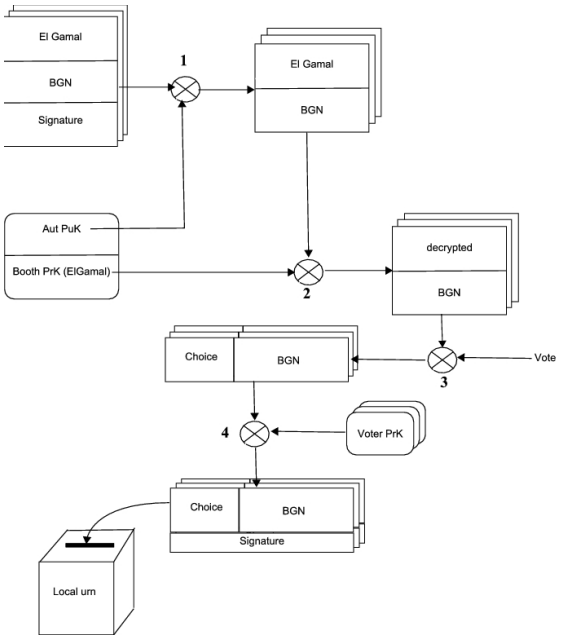

Fig. 2. Additional requirements do not require network during voting

In the top left of the figure we represent bulletin generation before the voting process starts:

1. The system uses generic bulletins which are permutations of the ordered list of candidates.
2. The bulletins are generated by mix-nets. The permutations are duplicated and each part is encrypted with a different algorithm:
   (a) One in ElGamal with a public key for each booth (`booth PuK`).
   (b) A second in BGN with the public key of the global urn.
3. After encryption, the bulletins are signed by a trusted authority.

During the voting process all communications between components are local:

1. The booth verifies the signature of the bulletin.
2. The booth decrypts the permutation encrypted in ElGamal.
3. The elector makes his/her choice and the decrypted permutation is destroyed.
4. The bulletin (the choice and the permutation encrypted in BGN) is then signed by the voter and put in the local urn.

Thus, the voting process does not depend on network communications.

We chose to use 2 different encryption schemes to meet 2 different requirements. BGN is required in order to provide a homomorphic mechanism for counting encrypted votes (as a whole) and decrypting the final result. This is a computationally complex algorithm and so we do not wish to use it to implement all our cryptolographic functionality. Thus, we chose to use El Gamal where we optimize the computation by not requiring a homomorphic technique.

A final aspect that should be noted is that voter authentication, in our chosen system, is carried out (indirectly) after the voting process has terminated. No person is refused permission to record a vote in an urn — but during the counting process (in the global urn) all non-authentic votes are rejected in a first step. This approach can be complemented by additional checks at voting stations that permit only people entitled to vote (over-18s, for example) access the voting booths. However, our approach is robust to any failures in this intitial filtering that would allow unauthorised voters access to the booth or urn. Our system would also be robust against someone authorised to vote being refused permission to vote at a particular station because this voter could try voting elsewhere.

5 Algebraic Specification

5.1 Specification and Validation of Count Rules

In previous work[12] algebraic techniques were used to model and validate the complex counting rules of the Irish parliamentary elections. A snippet of the algebraic specification of a `Vote` — a list of preferences for candidates — is given below:

```
--> *************************
--> defining the Vote module
--> *************************
mod! Vote {
  [Vote]
  protecting(NAT)
  protecting(ListNats)

--> some ops hidden

  op empty : Nat -> Vote
  op isempty : Vote -> Bool
  op addP : Vote Nat -> Vote
  op numCandidates : Vote -> Nat
  op invariant : Vote -> Bool
  op hasPref : Vote Nat -> Bool

--> some variables hidden
--> some equations hidden

  eq invariant(empty(numCs)) = false .
  ceq invariant(addP(v,n)) = true if (n <= numCandidates(v)) and
                                     (not(hasPref(v,n))) .
  ceq invariant(addP(v,n)) = false if (n <= numCandidates(v)) and
                                     (hasPref(v,n)) .
  ceq invariant(addP(v,n)) = false if n > numCandidates(v) .
}
```

A separate study[7] has shown the advantages of such formal models, over natural language descriptions, for the specification and validation of such algorithmic requirements. Motivated by their conclusions — and by the fact that requirements validation has been a major problem in voting systems — we chose to follow an algebraic approach to the specification of the data and data transformations in our e-voting system architecture. Further, in order to demonstrate that our approach is generally applicable, we chose not to develop a system that was appropriate only for the simplest type of counting algorithm (as with Presidential elections in France): our architecture has been designed to support the most complicated PRSTV voting schemes.

We note that these algebraic specifications in CafeObj were re-used because they provide a formal model that had already been validated to correctly represent the count algorithm. The transformation of the CafeObj models to an object oriented implementation language (like Java) follows well established methods[11]. The count implementation that results from the CafeObj specifications has an important role to play in later verification of the final prototype where rhe count operates on encrypted votes and involves a single decryption of the result.

This encrypted-count mechanism has not been formally verified and so we need some means of checking that it is correct. Our approach uses the

un-encrypted count (whose development was rigorously driven by the CafeObj models) as an oracle for testing the encrypted-count. In other words, we apply a form of regression testing to show that the results produced by the secure system are in agreement with the insecure system.

5.2 Verification of Data Transformations (Using Event-B Contexts)

A recurring reported problem with e-voting systems is the loss of votes arising from transport between system components; for example, from interface to urn, and from urn to count module. This problem is excacerbated by changing the way in which votes are represented as they move through the system. For example, with preferential voting, votes are typically recorded at the interface as an array of preferences.

In figure 3 we see how the way in which vote information is stored can change as votes move through the system.

In the interface, the voter conceptualises their vote as an array of candidates, some of whom are accorded preferences. However, in the ballot module the count algorithm "sees" each vote as an ordered sequence of preferences. As votes are transformed from one representation to another it is possible that a bug could transform a valid vote into an invalid vote[6]. Thus, we chose to specify such functions using Event-B contexts. In this way we formally verify (with the RODIN

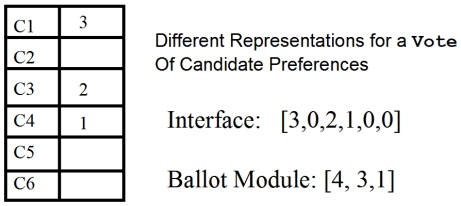

Fig. 3. Votes represented in different ways in the same system

```
CONTEXT
  ISARCS10
SETS
  set1
CONSTANTS
  numCandidates
  interface
  module
  move
AXIOMS
  axm1   :   numCandidates ∈ N1
  axm2   :   interface ∈ 1‥numCandidates ⟶ 0‥ numCandidates      //   a vote as stored at machine interface
  axm3   :   module ∈ 1‥ (card(ran(interface))−1) ⟶ 1‥ numCandidates      //   a vote as stored in the ballot module memory
             ∀i,j· i∈ 1‥numCandidates ∧ j ∈ 1‥
  axm5   :   numCandidates ∧ (interface(i) = interface(j)) ⇒                    //   only 0 as repeated element in interface
             ( (i = j) ∨ (interface(i) = 0)  )                                 //   where 0 represents no preference for that candidate
  axm6   :   ∀i,j· i↦j ∈ interface ∧ j>0 ⇒ j↦i ∈ module      //   the transformation from interface to module
  axm7   :   ∃ m· m ∈ N1 ∧ 0‥m ⊆  dom(interface)      //   cannot have missing elements in preference sequence
  axm4   :   move ∈ (1‥numCandidates ⟶ 0‥numCandidates) ⤖         //   to prove that the move transformation is a bijection
             (1‥ (card(ran(interface))−1) ⟶ 1‥ numCandidates)
END
```

Fig. 4. Proving theorems about the data in the e-voting context

tool) that such transformations are correct. A simplified context specification, in figure 4 illustrates how the RODIN tool is used to prove theorems about the election data, as specified in an Event-B context.

The main property that we wish to prove is that the move transformation is a bijection. We note that the algebraic specification (in Event-B) of the module corresponds to that which is used in the CafeObj specification of the Vote. The advantage of re-formulating the model in Event-B is that the invariant properties can be proven when we specify the dynamic properties of the system as a machine which executes events.

6 Refinement for Formal Verification of Design Steps (Using Event-B)

As a first step towards verifying our system to be correct, we abstract away from multiple voting locations. Our goal is to prove certain properties about this simple architecture and then to refine the architecture in order to add further details/components. If we can prove this refinement to be correct then all properties that we have proven for the initial simple architecture will be guaranteed to be correct for the more detailed architecture.

Our first refinement step is to offer 2 voting locations. (This will then be refined to an arbitrary number of voting stations). A simplified part of the machine specification, in figure 5 illustrates how a new event — for adding a new voting station – can be added to the architecture in such a way that the RODIN tool verifies the correctness of the design step as a refinement. Here, the Event-B is used to model refinement between abstract machines, where behaviour is partially specified by the shared context in which the machines operate.

```
add_iso   ≙
STATUS
  ordinary
ANY
  is
WHERE
  grd1   :   is ∈ isoloirs
THEN
  act1   :   isols ≔ isols ∪ {is}
END
```

Fig. 5. Modelling an architectural step as a refinement

7 The Prototype Implementation

As a proof of concept, we wished to implement this architecture as quickly as possible (following a rapid-prototyping development process). This implementation would replace the abstract network in our formal specifications with concrete communication protocols across the internet. Thus, we would be able to

demonstrate the feasibility of an implementation and validate the analysis from simulation of our formal models.

The underlying technology used to implement the distributed voting system prototypes — without any security mechanisms — was: Windows XP, Apache 2.2.9, MySQL 5.0.51, and PHP 5.2.6. On top of this, we built — using the httpunit tool that is popular in agile development of web sites[34] — an election simulation which simulated the behaviour of voters during the voting period (instantiating the same parameters as used in simulating our formal models).

We note that our final prototype — including the security mechanisms — is built on Java technology. The architecture of the final prototpye is consistent with that used in our initial simulation. This increases confidence that the quality of service requirements will continue to be met; but we have not yet been able to execute the same simulations on the final prototype.

7.1 Simulation: Validation of Formal Requirements Model

Although not a primary goal of the prototype development, it was clear that we could build a generic implementation that could be instantiated to all of our main architectural options (including the only one which we felt was feasible). Thus, we were able to simulate `VoteAnywhere` elections — using the internet as our underlying communication network — for all the options. In figure 6 we see that — for the purposes of simulation — we instantiated only two different polling stations, each with three polling booths. This was sufficient for simulating all different scenarios of interest.

It was no surprise that the architectures that failed to meet quality-of-service requirements when we simulated the formal models also failed to meet the requirements when implemented using a real network: it is necessary but not sufficient that the abstract models meet the requirements in order for the concrete implementations to meet them. (For more details of the simulation results see [13].)

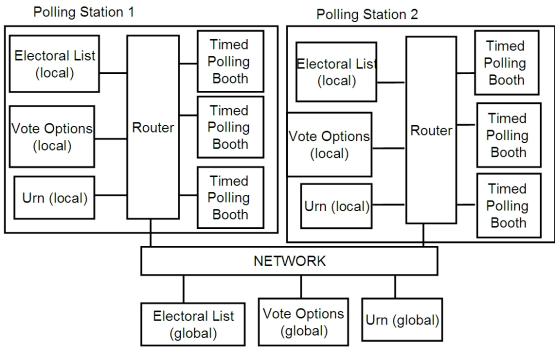

Fig. 6. Generic Architecture With 2 Polling Stations

We note that our formal models also abstracted away from communication time between system components, whether connected on a local or non-local network. We argued that such delays would be insignificant compared with the time taken for the elector to record a vote. Our election simulations validated the correctness of our abstraction — the speed of the internet connection (provided the service was available) had no effect, for all architectures tested, on the quality-of-service offered to the voter.

To conclude, our election simulation prototype demonstrated the feasibility of our architecture for meeting its requirements. The main outstanding concern was the implementation of the global clocks (see next section).

7.2 Trustworthy Global Clocks: Implementation Choices

In our simulations we made the assumption that the clocks on our different machines were synchronised; but we made no effort to guarantee that the assumption was met. We considered three implementation choices for ensuring that our clocks are synchronised in any further development of our distributed system:

1. **Network Time Protocol (NTP)**[26]: is a protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks. It would be the most appropriate solution to providing synchronised clocks between polling booths if we had a reliable network connection.
2. **Atomic Radio Clocks**[35]: the IEEE 1588 standard is designed for local systems requiring better accuracy than that provided by NTP. It is also designed for use where the cost of a GPS receiver in each communicating component is too high, or for where GPS signals are not reliable (or accessible)
3. **GPS Clocks**[32]: have already proven themselves in distributed real-time systems. We note that such a component could also facilitate automated verification of the location of voters (at particular polling stations). The potential impact (both positive and negative) of such information being available requires further analysis.

As none of these options is costly (with respect to the total cost of each voting booth and polling station) we propose that each booth have access to time generated by all three options (which may be controlled by a central machine in each polling station). Thus, the system would be robust against denial-of-service for any two of these three options during voting. Furthermore, the redundancy would introduce an extra level of security against some attacker attempting to manipulate the timestamp information on recorded votes (through manipulation of the local clocks).

7.3 Model Integration

The first prototype — without the security mechanisms — was developed by a software engineer with 4 years experience. The engineer was presented with all the different formal models that we had produced. The Estelle model was used

to construct the communication architecture. The algebraic specification (of a simplified count process) was used to develop the tabulation algorithm. The Event-B specifications of the design steps (refinements) played no role in the coding process — other than convincing us that the design was correct before it was implemented. The Event-B context specifications guided the implementation of Java code for specifying and verifying invariant properties. The engineer chose not to use extensions to the Java language that directly supported design by contract. Rather, they simply specified boolean invariant methods and threw runtime exceptions when such invariant properties were broken. These invariant properties helped identify coding errors (in the initial stages of implementation) but played little role in the verification of the design. Through documentation of the implementation code, we identified minor inconsistencies between the different models — these were mostly syntactic in nature.

The final prototype — with the security mechanisms — is in the process of being tested (against functional requirements). Through these tests (which were independently developed from the requirements models) we can verify that the count is correct, and that the three main features — *VoteAnywhere*, *Revote* and *Procuration* interact as required. We have no formal verification that the encryption algorithms central to the security mechanisms are correctly implemented — but the developers are experienced in using these same algorithms in a large number of security-critical systems.

It is clear from analysis of our development approach that the integration of our formal models is ad-hoc. We believe that are advantages from using different formal models at different stages of the development. However, establishing a re-usable method that coherently integrates such a mix of approaches is future research.

8 Conclusions

We have developed a prototype of an innovative voting system and addressed the major problem of denial-of-service attacks in a distributed architecture.

We have demonstrated that the development of an e-voting system can be done more rigorously through the use of formal methods, and that different modelling languages offer different advantages and disadvantages. The case study has not formally modelled all aspects and components of our voting system; a more complete model is work in progress. An important issue is a more formal integration of the different models that we have developed.

In current and future work we model our specific chosen system as a single member of a family of voting systems, where family members offer a unique subset of voting features[14]. We are also analysing the role of formality in maintaining these systems as requirements evolve (often due to changes in standards [15]).

Acknowledgements

Thanks to the anonymous reviewers for their comments and suggestions.

References

1. Abrial, J.-R., Butler, M.J., Hallerstede, S., Voisin, L.: An open extensible tool environment for event-b. In: Liu, Z., He, J. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 588–605. Springer, Heidelberg (2006)
2. Anane, R., Freeland, R., Theodoropoulos, G.: E-voting requirements and implementation. In: The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services. CEC/EEE, Tokyo, Japan, July 2007, pp. 382–392 (2007)
3. Braun, N., Brändli, D.: Swiss e-voting pilot projects: Evaluation, situation analysis and how to proceed. In: Krimmer [20], pp. 27–36
4. Bryans, J.W., Littlewood, B., Ryan, P.Y.A., Strigini, L.: E-voting: Dependability requirements and design for dependability. In: ARES 2006: Proceedings of the First International Conference on Availability, Reliability and Security, Washington, DC, USA, pp. 988–995. IEEE Computer Society Press, Los Alamitos (2006)
5. Cansell, D., Gibson, J.P., Méry, D.: Formal verification of tamper-evident storage for e-voting. In: Hinchey, M., Margaria, T. (eds.) Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2007), London, England, UK, pp. 329–338. IEEE Computer Society Press, Los Alamitos (2007)
6. Cansell, D., Gibson, J.P., Méry, D.: Refinement: A constructive approach to formal software design for a secure e-voting interface. Electronic Notes in Theoretical Computer Science 183, 39–55 (2007)
7. Carew, D., Exton, C., Buckley, J., McGaley, M., Gibson, J.P.: Preliminary study to empirically investigate the comprehensibility of requirements specifications. In: Romero, P., Good, J., Acosta Chaparro, E., Bryant, S. (eds.) Psychology of Programming Interest Group 17th annual workshop (PPIG 2005), pp. 182–202. University of Sussex, Brighton (2005)
8. Chaum, D., van der Graaf, J., Ryan, P.Y.A., Vora, P.: Secret ballot elections with unconditional integrity. Report CS-TR-1058, Department of Computing Science, University of Newcastle upon Tyne (2007)
9. Chen, Y.-Y., Jan, J.k., Chen, C.-L.: The design of a secure anonymous internet voting system. Computers & Security 23(4), 330–337 (2004)
10. Clarkson, M.E., Chong, S., Myers, A.C.: Civitas: A secure remote voting system. In: Chaum, D., Kutylowski, M., Rivest, R.L., Ryan, P.Y.A. (eds.) Frontiers of Electronic Voting. Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), vol. 07311, Schloss Dagstuhl, Germany (2007)
11. Gibson, J.P.: Formal Object Oriented Development of Software Systems Using LOTOS. Thesis CSM-114, Stirling University (August 1993)
12. Gibson, J.P.: E-voting requirements modelling: An algebraic specification approach (with cafeobj). Report NUIM-CS-TR-2005-14, Department of Computer Science, National University of Ireland, Maynooth (2005)
13. Gibson, J.P., Lallet, E., Raffy, J.-L.: Analysis of a distributed e-voting system architecture against quality of service requirements. In: The Third International Conference on Software Engineering Advances (ICSEA 2008), pp. 58–64. IEEE Computer Society Press, Los Alamitos (2008)
14. Gibson, J.P., Lallet, E., Raffy, J.-L.: Feature interactions in a software product line for e-voting. In: Nakamura, Reiff-Marganiec (eds.) Feature Interactions in Software and Communication Systems X, Lisbon, Portugal, June 2009, pp. 91–106. IOS Press, Amsterdam (2009)

15. Gibson, J.P., McGaley, M.: Verification and maintenance of e-voting systems and standards. In: Remenyi, D. (ed.) 8th European Conference on e-Government, Lausanne, Switzerland, July 2008, pp. 283–289. Academic Publishing International (2008)
16. Hoffman, L.J.: Internet voting: will it spur or corrupt democracy? In: CFP 2000: Proceedings of the tenth conference on Computers, freedom and privacy, pp. 219–223. ACM, New York (2000)
17. ISO/IEC. Estelle: A formal description technique based on an extended state transition model. Technical Report ISO 9074, Information technology - Open Systems Interconnection (1997)
18. Jefferson, D., Rubin, A.D., Simons, B., Wagner, D.: Analyzing internet voting security. ACM Commun. 47(10), 59–64 (2004)
19. Joaquim, R., Zuquete, A., Ferreira, P.: REVS — A Robust Electronic Voting System. In: Proceedings of the IADIS International Conference on e-Society, Lisbon, Portugal, June 2003, pp. 95–103 (2003)
20. Krimmer, R. (ed.): Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, Castle Hofen, Bregenz, Austria, August 2-4. LNI, vol. 86. GI (2006)
21. Krimmer, R., Triessnig, S., Volkamer, M.: The development of remote e-voting around the world: A review of roads and directions. In: Alkassar, A., Volkamer, M. (eds.) VOTE-ID 2007. LNCS, vol. 4896, pp. 1–15. Springer, Heidelberg (2007)
22. Maaten, E.: Towards remote e-voting: Estonian case. In: Prosser, A., Krimmer, R. (eds.) Electronic Voting in Europe. LNI, vol. 47, pp. 83–100. GI (2004)
23. Madise, Ü., Martens, T.: E-voting in estonia 2005. the first practice of country-wide binding internet voting in the world. In: Krimmer [20], pp. 15–26 (2005)
24. McGaley, M., Gibson, J.P.: E-voting: a safety critical system. Report NUIM-CS-TR-2003-2, Department of Computer Science, National University of Ireland, Maynooth (2003)
25. McGaley, M., Gibson, J.P.: A critical analysis of the council of europe recommendations on e-voting. In: EVT 2006: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop, pp. 9–22. USENIX Association (2006)
26. Mills, D.L.: Internet time synchronization: the network time protocol. IEEE Transactions on Communications 39(10), 1482–1493 (1991)
27. Qadah, G.Z., Taha, R.: Electronic voting systems: Requirements, design, and implementation. Comput. Stand. Interfaces 29(3), 376–386 (2007)
28. Roth, S.K.: Disenfranchised by design: voting systems and the election process. Information Design Journal 9(1), 1–8 (1998)
29. Rubin, A.D.: Security considerations for remote electronic voting. ACM Commun. 45(12), 39–44 (2002)
30. Sandler, D.R., Wallach, D.S.: The case for networked remote voting precincts. In: EVT 2008: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop, Berkeley, CA, USA, July 2008. USENIX Association (2008)
31. Selker, T., Goler, J.: The save system — secure architecture for voting electronically. BT Technology Journal 22(4), 89–95 (2004)
32. Sterzbach, B.: Gps-based clock synchronization in a mobile, distributed real-time system. Real-Time Syst. 12(1), 63–75 (1997)
33. Storer, T., Duncan, I.: Practical remote electronic elections for the uk. In: PST, pp. 41–45 (2004)

34. Tappenden, A., Beatty, P., Miller, J.: Agile security testing of web-based systems via httpunit. In: ADC 2005: Proceedings of the Agile Development Conference, Washington, DC, USA, pp. 29–38. IEEE Computer Society Press, Los Alamitos (2005)
35. Weibel, H., Béchaz, D.: IEEE1588 Implementation and Performance of Time Stamping Techniques. In: Conference on IEEE 1588, Gaithersburg (september 2004)
36. Zúquete, A., Almeida, F.: Verifiable anonymous vote submission. In: SAC 2008: Proceedings of the 2008 ACM symposium on Applied computing, pp. 2159–2166. ACM, New York (2008)