Feature Interactions in a Software Product Line for E-voting

J Paul GIBSON [a], Eric LALLET [a], Jean-Luc RAFFY [a],
[a] *Telecom & Management SudParis (TMSP),*
(UMR 5157 INT-CNRS SAMOVAR),
9 rue Charles Fourier,
91011 Évry cedex, FRANCE

Abstract. A significant number of failures in e-voting systems have arisen because of poorly specified requirements, combined with an ad-hoc approach to engineering multiple variations of similar machines. We demonstrate that e-voting is a suitable domain for leveraging state-of-the-art in software product line (SPL) engineering techniques and tools. We propose, based on examples of typical requirements, that a feature-oriented approach to e-voting domain analysis is a good foundation upon which to carry out commonality and variablity analysis. Simple analysis of our core and optional features (and their variants) leads us to believe that feature interactions are a major problem in voting systems. We conclude that a formal software product line would help to manage the composition of features in such a way as to eliminate interactions in the requirements models, before particular e-voting systems are instantiated.

Keywords. E-voting, Feature Interactions, Software Product Line, Requirements, Domain Modelling

1. Introduction

The software in e-voting machines has not, in general, been well-engineered[1]. Many governments have chosen to adopt e-voting as a show-case for innovative technology[2]. It is a poor reflection on the profession of software engineering that the software in these systems is, in general, neither trusted nor trustworthy[3]. We propose that the software engineering community should look upon this as an opportunity to demonstrate just how much software engineering methods, techniques and tools have evolved since the turn of the century[4]; and that the software industry is now mature enough to develop e-voting machines that are highly dependent on software and that are highly dependable.

Software Product Lines (SPLs) [5] are attracting attention in the area of applied software engineering research. The challenge, which this article addresses, is to demonstrate how and why an e-voting SPL could be built. E-voting systems correspond in terms of size and complexity to those reported in a number of SPL case studies [6]. The number of variations across systems[7] is large enough to

merit an SPL approach, but not so large as to be unmanageable. Furthermore, these systems exhibit a large amount of common functionality and so the potential for re-use is high. The aspect of e-voting that may be more challenging is that the software may be considered (safety or mission) critical [8]. However, recent research suggests that SPLs can be used to develop safety critical systems [9].

Many of the problems that have arisen in the domain of e-voting have arisen because of poorly specified requirements and standards documents [10]. It has been proposed that a comprehensive domain analysis be carried out before standards are re-engineered [1]. The resulting domain models should provide the foundations upon which standards could be built; and they would also play a major role in the development of an e-voting SPL.

We note that analyses of existing systems — such as the state of Ohios EVEREST report[11] — have identified verification issues directly related to foundational concepts in software product lines: "parameterized families of components" [12], a "family of related programs" [13,14], and "structuring commonality and variability according to features"[15].

2. Background and Motivation

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 in France. 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 French electorate. More importantly, through the development of the prototype, we wish to establish best practice in the engineering of e-voting systems, with emphasis on software quality. This paper proposes that such best practice corresponds to the synthesis and analysis of a generic framework for the development of e-voting systems that can meet a range of different, yet well-understood, requirements; in other words, a product line.

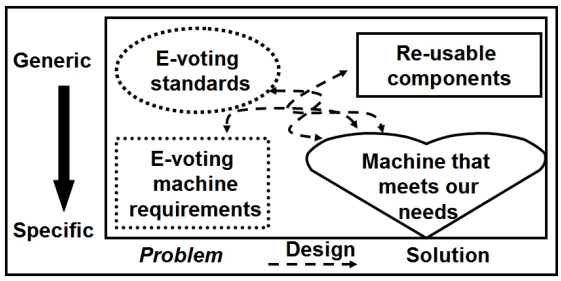

Figure 1. The Re-use Knot

In figure 1, we see a graphical representation of our main problem, that of re-use: we know that requirements modelling is critical in the production of quality systems, we understand that our system must meet certain standards and we wish to re-use particular components in order to provide innovative functionality.

In effect, there is a complex knot of lines of re-use that must be managed during design. In the next section we report on our initial efforts to untangle this re-use knot through domain modelling.

3. Domain Modelling

Through initial analysis of the standards — both French[16] and European[17] — we identified that some of the chosen system requirements were not addressed, some were partially addressed, and some were completely addressed. Also, many of the standards were not applicable to the chosen system. This can be seen in the top left of figure 2.

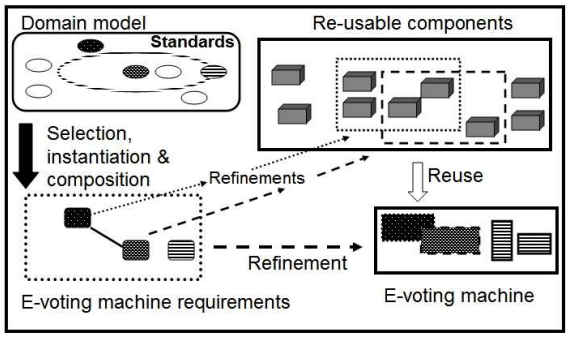

Figure 2. Domain Modelling and Standards

In order to move from generic standards to specific requirements we need a complex process of choosing the standards that are applicable, instantiating them to the parameters of our chosen system and then composing them with the requirements that are specific to our chosen system. This is represented in the left hand side of figure 2. We understand certain of our requirements well-enough that we can start to map them to particular re-usable components. This can be seen in the top right of figure 2.

Our research project has yet to develop a prototype, but we show how such a machine would fit into our framework (in the bottom right of figure 2). This framework has helped us to improve on our initial situation: the reuse knot has been untangled. However, we have identified two key problems that motivate us to try and improve the framework. Firstly, the mapping between the domain model and the re-usable components is not clear. Secondly, we know what we should re-use and what we could re-use but we do not have good understanding of how to re-use: in other words, selection and instantiation would be well-managed (perhaps with automated tool support) but composition would be ad-hoc.

3.1. Problems With Terminology

In our first project meeting, project members were using different words to mean the same thing, and using the same word to mean different things. A main prob-

lem was that the definition and usage of terms was not consistent within and between standards documents. Furthermore, the European standards were written in English, whilst the French regulations were not. In order to address this issue, we examined whether domain engineering (for software product lines) had specific approaches to help with the problem of terminology. We identified two candidates: a generic approach based on ontologies [18], and specific re-use of the Election Markup Language (EML) of the *Organization for the Advancement of Structured Information Standards*[19]. Finally, we chose to formulate our own key terminology, which is used consistently throughout the remainder of this paper. The following list is intended as a guide to the reader.

Authentication: legal means by which an **elector** identifies themselves in order to be allowed to record their **individual vote**.
Ballot: legal means by which an **elector** records their **individual vote** at an **election**.
Candidate: a person (or list of persons) for whom an **elector** can make a choice at a specific type of **election**
Candidate List: a list of **candidates** making up the **option list**
Count: process by which **individual votes** of **voters** give rise to a global **result**.
Elector: an individual member of the **electorate**
Electoral List: the legal means of recording the **electorate**.
Electorate: the **electors** entitled to an **individual vote** at an **election**.
Election: process by which an **electorate** collectively make a choice of one or more options from an **option list**.
Election Type: categorises the information that an **elector** is expected to provide on a **ballot** in order to **record an individual vote**.
Individual vote: expression of choice from **option list** on a **ballot**.
Option List: the possibilities presented to each **elector** at an **election** — most commonly a **candidate list**.
Polling Booth: area in **polling station** where an **elector** can **record their individual vote** on a **ballot** (usually in secret).
Polling Station: location where an **elector** can go to **record their individual vote**, perhaps using a particular **Polling Booth:**
Record an individual vote: when a **voter** places their **individual vote** in the **urn**.
Result: the collective wish of the **electorate** as decided by the **count** of their **individual vote**.
Urn: where **individual votes** are stored before the **count**; also known as the **ballot** box.
Vote: the collection of **individual votes** from **voters**.
Voter: an **elector** who has recorded (or is in the process of recording) their **individual votes**, using a **ballot**.
Voter List: the offical means of recording the **voters** (a subset of the **electoral list**).
Voting System: the means by which the **election** is implemented.

3.2. Variability in Mandatory Features: the Example of Election Type

The core functionality of any e-voting system must include four mandatory features: election set up, recording individual votes, the count and returning a validated result. (These can be seen in figure 3, when we consider an example of the requirements of a specific e-voting system).

The main variability between all e-voting systems is the election type[20]:

I. A single option from 2 possible options.
II. A single option from n possible options (setting n to 2 gives type *I*).
III. A minimum number of options, m, from n possible options, where $m \leq n$ (setting $m = 1$ gives type *II*).

For type *III* we may also distinguish by types of ordering (for preferential voting):

i. All options must be ordered.
ii. No ordering exists betwen options.
iii. Options may be (partially) ordered.

The variability over what information can be recorded on a ballot will have impact over the variability of core functionality. In particular, the election set up must finalise the option list and specify which of the particular election types is being used. This will have a consequence on how an elector interacts with the voting system in order to record their individual vote. The ballot must be consistent with the option list and election type. The urn must be consistent with the storage of ballots. The count process must be consistent with the ballots. Finally, the result of the election must be consistent with the counting of the ballots.

3.3. Adding Optional Features

In this section we introduce some of the most common optional features of existing e-voting systems.

Voter Registration: a person can apply to be placed on the electoral list.
Voter Anonymity: the individual vote of a voter is not made public.
Voter Authentication: to be given a ballot, a person is identified as an elector on the electoral list.
Voter Authorisation: a person with a ballot must be permitted to record their individual vote.
Voter Verification: a voter can verify that their individual votes was correctly recorded and counted.
Procuration: an elector may allow another person to record a individual vote on their behalf.
Obligation to vote: non-voters may be prosecuted.
Re-count: under certain well-defined circumstances, the count process has to be re-executed.

Adding such optional features to the existing core architecture is a non-trivial problem and gives rise to the type of interactions that typically occur when com-

posing requirements[21]. In figure 3, we illustrate how a high-level object-oriented view of the requirements of a particular voting system, including optional features, can aid the process of identifying how features can be implemented using re-usable components, and how these features can be composed in order for a voting system to meet its requirements. The model in figure 3 is not intended to be generic; it is an example of how a requirements model for a specific voting system can be engineered re-using a core architecture, together with feature increments.

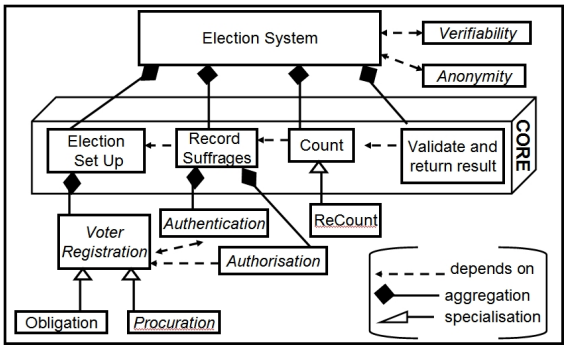

Figure 3. Adding Features To Architecture

In the diagram we see that we have introduced elector authentication and authorisation features. We do not categorise these as core features because a minority of elections do not have these requirements. In the figure we show the feature composition for authentication, authorisation and registration as simple aggregation relationships with a single core feature. In the model we also see features that are introduced as specialisations of existing features: procuration and obligation are specialisations of the registration process, whilst we chose to model recounting as a specialisation of the count feature. Of course, each of these optional features are open to specialisation themselves.

Two other optional features of interest are voter anonymity and verifiability. It is not clear, due perhaps to our lack of domain understanding, how such features could be cleanly plugged into a core architecture. In our model in figure 3 we have represented these features as being interdependent with the whole voting system. In the next subsection we show how anonymity is a much more complex feature, with many variants, than one would initially suspect. We believe that aspects may hold the key to modelling e-voting features, like anonymity, that cut across multiple components of the architectural core.

3.4. Variability in Optional Features: the Example of Anonymity

We report on our analysis of the anonymity feature, as it is a good example of the levels of variability that exist in all the optional features:

1. All individual votes are anonymous and anonymity is enforced (thus the system is coercion free).

2. All individual votes are transparent (not anonymous) and transparency is enforced.
3. A mix of anonymity and transparency is permitted.

Within types 2 and 3, we must consider degrees of transparency, for example:

i. Anyone can request and be permitted to see the individual vote of any elector.
ii. Only specified election officials may request and be permitted to see the individual vote of any elector.
iii. An elector can choose to permit (or not) some other person. to see their individual vote.

Within type 3, we must also consider whether the anonymity type of an individual elector is:

a. A choice open to all electors.
b. A choice open to some electors.
c. Enforced by the status of elector.

Finally, we can classify the fact that an elector has registered an individual vote (without revealing the individual vote) as information that is:

I. Open to the public.
II. Available to authorised officials.

Let us now consider a specific anonymity requirement — all individual votes are anonymous (type 1) and whether an elector has voted is public (type I). We note that these requirements are common to the Irish and French systems of voting, yet their implementations are different in rather subtle ways. We examine this in more detail in section 5, when new voting features are introduced.

4. An SPL for e-voting

We have decided that our domain model should not be structured around the poorly engineered standards documents. We will — where possible — try to validate our domain model against current standards, but the domain model now plays a more important new role: as a requirements specification for our software product line. The goal is to have a direct mapping between components in the SPL and the features (requirements increments) in the domain model. This is illustrated in figure 4.

There are obvious advantages with this framework when compared with that in figure 2. Firstly, verification of the correctness of refinements — a good example is the refinement of the voter interface[22] — can be done at the generic level (facilitating re-use). Secondly, new components and features can be soundly integrated into the framework through the process of abstraction into the domain model. Thirdly, we should be able to formalise different composition mechanisms in order to automate the configuration process. In the sections that follow we continue with our feature oriented approach to modelling commonality and variability, similar to that outlined in [23].

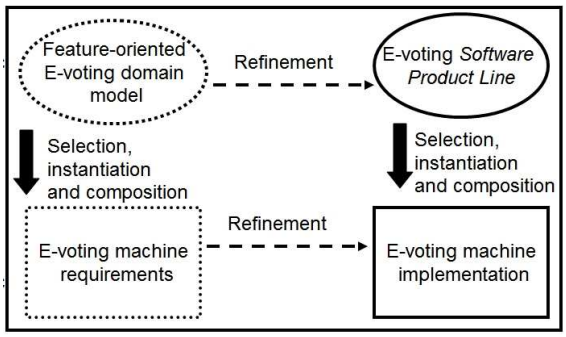

Figure 4. The E-voting Feature-oriented SPL

5. New Requirements as Features

In our chosen system, we introduce two clear voting innovations (for France) that the system will be required to support.

Firstly, we wish to allow voters to be able to go to any polling station in order to record an individual vote: currently they are required to go to a particular station. We refer to this as a *VoteAnywhere* feature. This feature is aimed at addressing the problem of electors who wish to register an individual vote not doing so because the polling station at which they are registered to vote is not easily accessible to them whilst the election is open.

Secondly, we wish to allow electors to be able to re-vote so that a previously recorded individual vote is overwritten by a new individual vote: currently electors have no way of changing their individual vote whilst the election is open. We refer to this as a *ReVote* feature. Such a feature is more important when electors can vote over a long period of time, as with "early voting" in many American states. Furthermore, the notion of allowing an elector to re-record an individual vote is one which often arises when considering remote electronic voting: one could argue that allowing a *ReVote* could hinder the buying/selling of votes in this situation.

5.1. Vote Anywhere

The fundamental requirement of *VoteAnywhere* is that any location (and/or mechanism) for recording an individual vote which is usable by one elector must by usable by all electors. Variants on this feature will include voting at any polling booth, internet voting, telephone voting, postal voting etc. We note that all such variants will require rigorous definitions of "usable", "location" and "mechanism". Further, it is likely that any *VoteAnywhere* feature will incorporate well-defined exceptions in practice.

Before we analyse the new feature in detail, we should fully understand how the current system functions in the context in which our chosen system is expected to be used. Currently in France, each voter is assigned to a unique polling station where their name is found on an electoral list. In order to vote, their identity is authenticated and consequently they may be authorised to place a ballot in the urn. Checking that someone has not already voted involves a simple protocol:

just after a voter is allowed to place a ballot in the urn they are required to sign next to their name on the voter list. If they have already signed then they are considered to have already registered their individual vote and will not be allowed to place the ballot in the urn. In the exceptional case where a voter refuses to sign, an election official signs on their behalf and a public announcement of this is made.

The *VoteAnywhere* feature introduces two potential problems. Firstly, the system must ensure that each elector records their individual vote using the correct ballot. In French local elections there are thousands of different candidate lists and this new requirement cannot be easily fulfilled using traditional paper ballots. However, an e-voting system offers the possibility of dynamically generating a specific ballot for each elector. Secondly, the system must ensure that an elector cannot record multiple individual votes (by going to different polling stations) and have their individual vote counted multiple times. Currently, this is managed by a single voter list at each polling station. This implementation guarantees the fundamental "1-person, 1-vote" requirement by ensuring that an elector cannot place more than one completed ballot in the urn.

5.2. Re-vote

The fundamental requirement of the *ReVote* feature is that an elector is permitted to record multiple individual votes, and if they chose to do so then only the last one is counted. This new feature poses many problems for the current French voting system: after an elector drops a ballot into the urn then there is no way of accessing the contents of the urn until the election process is closed. This mechanism is used to ensure that no-one can surreptitiously add or remove bulletins. We note that the urn is transparent. Transparency means that, in order to meet the additional requirement of secrecy, French voters are required to place their ballots in envelopes before they place them in the urn.

The current system could support re-voting provided there was a means of finding the ballot of a particular voter in the urn without disclosing the individual vote recorded on any of the ballots. We refer to such a ballot as being signed (in confidence). We note that this requires the elector to trust this signature process not to compromise the level of anonymity expected.

With trusted and trustworthy signatures, we have two obvious choices of protocol for meeting the *ReVote* requirement:

1. Before an elector is permitted to place a ballot in the urn, they are required to sign it so that it can uniquely identify them as the voter. Then all previous ballots in the urn that share this signature (there should be only one) must be destroyed. Finally, the new (signed) ballot must be placed in the urn.
2. Timestamp the ballots and when the recording of individual votes is terminated one must destroy all ballots except the most recent ballot of every voter.

Option (1) would not be feasible with traditional systems, except where the number of electors was small. Option (2) is more feasible but would require a signif-

icant additional resource for filtering re-votes before counting takes place, with increased possibility of human error. Furthermore, it would add much complexity to any auditing process.

5.3. ReVote and Anonymity: some design lessons

In the current French voting system, anonymity is achieved by placing ballots into sealed envelopes before they are placed in a transparent urn. The *ReVote* feature is easy to implement by adding another envelope for recording signatures, and into which the original sealed votes can be placed. Thus, we could put in place a process that guarantees that finding and destroying (or counting) ballots can be done without compromising anonymity.

Now, consider the Irish voting system whose design for ensuring anonymity does not use envelopes but does use an opaque urn. Imagine the situation where the Irish electorate request a *ReVote* feature, as it has proven so popular with their imaginary French neighbours. The temptation could be to directly re-use the *ReVote* component that has proven to work so well in the French system — i.e. just put the Irish bulletins inside signed envelopes. However, more rigorous analysis would show that this could introduce problems when it comes to the count process. In order to count a ballot, in this imaginary Irish election, it must be removed from the single signed envelope. If this is done by a human being then anonymity is compromised unless we can trust the person concerned not to link the signature with the individual vote on the bulletin. We acknowledge that there are ways around this potential interaction, but the example should illustrate the problems that we face in developing an e-voting SPL for component re-use.

6. SPL: Feature Interactions

An e-voting system has a myriad of layers of inter-related legal requirements to meet. Further, each voting system has to meet specific needs which are not directly addressed by the laws and standards. The requirements of the system must somehow integrate these specific needs with multiple layers of laws and standards. As changes are made to requirements within different layers, in parallel, then who is responsible for ensuring that the requirements can be re-integrated in a coherent manner?

In the following, we have selected examples of interactions that illustrate the need for more formal modelling and analysis. First we consider the most challenging requirements integration problem in e-voting: how to ensure both anonymity and verifiability? Secondly, we analyse potential interactions between the two innovative features of our chosen system: re-voting anywhere. Finally, we discuss the interactions that arise when the innovative features of our chosen system have to integrate with an existing non-core feature common to French elections: procuration.

In all instances we consider quality of service (*QoS*) to be a core requirement, so that the time required to record an individual vote should never be "unreasonable"[24]. In practice, as we cannot guarantee the reliability of any non-

local communication network, we currently reject any voting process where an elector depends on a non-local communication in order to be able to register their individual vote. This issue is critical in many of the feature interactions that we have considered.

6.1. Anonymity and Verifiability

This the classic example of requirements that appear to be contradictory — how can an elector's ballot be kept secret when we wish the elector to be able to verify that it has been correctly counted? It would appear that verifiability requires a voter to be able to follow their ballot through the count process (at the very least) but how can they do that without signing it, and if they sign it then how can it be kept anonymous? Recent research in cryptographic e-voting protocols[25,26,27] suggests that these two requirements can be met, provided we refine the notion of verifiability and we require the electors to follow specific verification procedures. However, it is clear that there are a number of subtle interactions between anonymity and verifiability when they integrate with other voting features, which results in different protocols making different compromises between competing criteria[28]. Another major problem is that many of these schemes depend on a reliable non-local network during the time in which an elector records their individual vote and so the *QoS* feature is compromised.

6.2. VoteAnywhere and ReVote

Recall that *VoteAnywhere* (in our chosen system) permits an elector to vote at any polling station and so the main issue is meeting the core requirement that an elector can only have a single individual vote counted. If an elector is to be precluded from re-voting then the only way to implement this is to have a voter list shared between polling stations. Consequently, this would require a network with a reliable quality of service.

Recall that a fundamental requirement of *ReVote* is that only the last individual vote recorded on a ballot will be counted. Now, provided that an elector has to record an individual vote at the same polling station then there should be no problem in identifying which individual vote was the last recorded when a *ReVote* occurs. Some obvious options are:

(a) Use a local clock to stamp each signature and ensure that the clock's time advances at a reasonable rate (so that two ballots from the same elector are guaranteed never to have the same timestamp). This would require a single clock for each polling station.
(b) Use a local counter to stamp each signature. This would require a counter for each elector.
(c) Use a "destructive-write" so that a signed bulletin added to the local urn automatically results in the destruction of any bulletin that shares the same signature already in the urn. This would require a special type of urn that facilitates such a "destructive-write". Further, with paper ballots the core *QoS* requirement may be difficult to guarantee as a *ReVote* would require searching the urn for a signed ballot.

Integrating *VoteAnywhere* with *ReVote* poses problems in all three of the optional designs above:

(a) If we continue to use local clocks at each polling station then these clocks would need to be synchronised to ensure that the correct ordering is maintained when an elector re-votes at different polling stations. In reality, polling stations are far enough apart that this would not require the use of expensive high-precision clocks. As an alternative, we could replace local clocks with a central global clock. However, local polling stations would need to be able to communicate with this global clock using some sort of network. Consequently, this would require a reliable non-local network in order to meet our core *QoS* requirement for each elector.
(b) When an elector can *VoteAnywhere* then counting the number of times an elector has already voted requires this data to be shared amongst voting stations. Consequently, this would also require a reliable non-local communication network.
(c) Using a "destructive-write" would require a shared global urn. Again, this would require a reliable non-local communication network.

We note that option (a), with local clocks, permits electors to *ReVote Anywhere* without requiring a reliable non-local network. This insight arose from analysis of simulations of different voting architectures for quality of service[24].

6.3. Procuration and ReVote

Procuration is the feature that permits one elector (known as the elector-by-procuration) to record an individual vote on behalf of another elector. In France, procuration does not necessarily prohibit an elector from recording an individual vote. In such a case, the elector can go to a polling station and record an individual vote, provided that the elector-by-procuration has not already done so. The main requirement for the *Procuration* feature is that an elector will be prohibited from recording an individual vote if this has already been done on their behalf.

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 an individual vote whilst in the second instance an elector must never be denied the right to record an individual vote. Consequently, to provide the *ReVote* feature in French elections it would be necessary to change the existing regulations with respect to *Procuration*. Further analysis reveals that *Procuration* currently does not allow a second vote the possibility of taking priority over the first simply because the paper system does not facilitate it. Given an individual vote recorded by *Procuration* this could be overwritten by a second individual vote through using a constrained instance of the *ReVote* feature. However, this would mean that the last individual vote would always have priority. This may not match our intuition of how *ReVote* and *Procuration* should work together: do we really want an individual vote by *Procuration* to take priority over one already recorded by the original elector?

6.4. Procuration and VoteAnywhere

Given a reliable non-local communication network then there are no undesirable interactions between *Procuration* and *VoteAnywhere*: an elector will be refused the right to record an individual vote if the elector-by-procuration has already done so on their behalf. Similarly, the elector-by-procuration will be refused the right to record an individual vote if the elector has already done so. The central voter list used to *VoteAnywhere* will guarantee that the elector and the elector-by-procuration cannot record two individual votes "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.

6.5. Procuration, ReVote and VoteAnywhere

Using local clocks to implement *ReVote Anywhere* can lead to timing interactions when combined with *Procuration*. Firstly, we now require much more precise clocks to ensure that the correct vote order is recorded when individual votes are submitted for the "same elector" (but, possibly two different people!) at different polling stations: the distance between polling stations is no longer a factor as an elector can now be considered to be in "two places at once". Another issue is that when an elector records an individual vote, for a final time, they have no guarantee that the individual vote will be counted. No centralised voter list (or urn) means that they must trust that an elector-by-procuration does not vote after them at a different polling station.

7. Conclusions and Future Work

The next generation of electronic voting systems should be better engineered than the current generation. A first step towards this is a more thorough and precise analysis of the voting domain. Then, procurement offices can leverage the understanding in such a model in order to better specify their requirements. Consequently, manufacturers should be encouraged to develop a SPL for e-voting machines, in order to best manage the obvious commonalities and variations.

In this paper, we have shown that a feature-oriented approach can be applied to the design of an e-voting SPL architecture. We believe that refinement has a key role to play in the development and application of such a SPL, particularly in the re-use of verified feature components. Integrating SPL techniques with formal methods is a promising approach: refinement for re-use of trustworthy components has already been addressed with respect to e-voting machine interfaces [22] and storage [29].

A major problem with e-voting systems is that they need to be trustworthy and trusted [30]. There are two key properties of SPLs that this paper has not directly addressed, but which must be examined if e-voting machines developed using SPLs are to be trusted: late binding and openness. Expecting electors to trust a voting machine has, until now, required them to trust that they have been properly verified to do what they are supposed to do. Thus, they trust the

machines indirectly through their trust of the agents who have verified them. This trust is dependent on the electors knowing that the machines they use to record their individual vote are precisley the machines that have been verified. In a SPL, late binding of components opens up the question of whether the precise system in front of the users has ever been verified. Furthermore, leaving a variant open to later specialisation begs the question of whether trust can be compositional: if the delivered system is trusted and the specialised variant is trusted then can the new system that binds the new variant to a specific feature be trusted without having to reverify the whole system? Expecting users to trust the system in this way is unreasonable if we cannot guarantee that our verification mechanisms are compositional. This returns us to developing an SPL specific to voting systems that uses composition mechanisms that can guarantee absence of unwanted feature interactions (before the system is delivered). Current research — based on the notion of a feature interaction algebra[31] — suggests that a *correct-by-construction* approach to guaranteeing the functionality of e-voting systems[22,29] merits further investigation.

Building an e-voting system has a high risk of failure due to unstable standards [1] and lack of understanding of the problem domain. Requirements creep has been a major problem in e-voting systems. A good example is of the requirement for a voter verifiable audit trail (VVAT) for increased security[32]. Many current e-voting machines do not meet this requirement, and were not designed to do so. However, the election administrators and manufacturers seem to believe that this additional functionality can be somehow bolted on to already procured machines without risk. An e-voting SPL should be developed in order to manage the risk of evolving requirements: current research on maintaining SPLs [33] suggests that developing an e-voting SPL that can evolve as standards change is feasible using current techniques, but that it is a non-trivial problem. This is current and future research.

References

[1] J. Paul Gibson and Margaret McGaley. Verification and maintenance of e-voting systems and standards. In Dan Remenyi, editor, *8th European Conference on e-Government*, pages 283–289. Academic Publishing International, July 2008.

[2] Jrgen Svensson and Ronald Leenes. E-voting in Europe: Divergent democratic practice. *Information Polity*, 8(1):3–15, 2003.

[3] Brian Randell and Peter Y. A. Ryan. Voting technologies and trust. *IEEE Security and Privacy*, 4(5):50–56, 2006.

[4] Gilda Pour, Martin L. Griss, and Michael J. Lutz. The push to make software engineering respectable. *IEEE Computer*, 33(5):35–43, 2000.

[5] P. Clements and L. Northrop. *Software product lines*. Addison-Wesley Boston, 2002.

[6] Jan Bosch. Product-line architectures in industry: a case study. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 544–554, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[7] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers & Security*, 25(2):137–153, 2006.

[8] Margaret McGaley and J. Paul Gibson. E-Voting: A Safety Critical System. Technical Report NUIM-CS-TR-2003-02, NUI Maynooth, Computer Science Department, 2003.

[9] Jing Liu. Handling safety-related feature interaction in safety-critical product lines. In *ICSE Companion*, pages 85–86. IEEE Computer Society, 2007.

[10] Margaret McGaley and J. Paul Gibson. A critical analysis of the Council of Europe recommendations on e-voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, pages 1–13, Berkeley, CA, USA, 2006. USENIX Association.

[11] Kevin Butler, William Enck, Harri Hursti, Stephen McLaughlin, Patrick Traynor, and Patrick McDaniel. Systemic Issues in the Hart InterCivic and Premier Voting Systems: Reflections on Project EVEREST. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, July 2008. USENIX Association.

[12] D. McIlroy. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98, 1968.

[13] Edsger W. Dijkstra. *Structured programming*, chapter Notes on structured programming, pages 1–82. Academic Press Ltd., London, UK, 1972.

[14] David Lorge Parnas. On the design and development of program families. *IEEE Trans. Software Eng.*, 2(1):1–9, 1976.

[15] K.C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU-SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[16] Ministère de L'Intérieur. Règlement technique fixant les conditions d'agrément des machines à voter. NOR : INTX0306924A, November 2003.

[17] Council of Europe. Recommendation on legal, operational and technical standards for e-voting. Rec(2004)11, September 2004. Adopted by the Committee of Ministers on 30 September 2004 at the 898th meeting of the Ministers' Deputies.

[18] Ricardo de Almeida Falbo, Giancarlo Guizzardi, and Katia Cristina Duarte. An ontological approach to domain engineering. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 351–358, New York, NY, USA, 2002. ACM.

[19] Government Technology. E-Vote: Election Markup Language 5.0 Approved as OASIS Standard, January 2009.

[20] Jonathan Levin and Barry Nalebuff. An introduction to vote-counting schemes. *Journal of Economic Perspectives*, 9(1):3–26, Winter 1995.

[21] J. Paul Gibson. Feature requirements models: Understanding interactions. In Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors, *Feature Interactions in Telecommunications Networks IV, (FIW 1997)*, pages 46–60. IOS Press, June 1997.

[22] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electr. Notes Theor. Comput. Sci.*, 183:39–55, 2007.

[23] Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *WICSA*, pages 45–54. IEEE Computer Society, 2001.

[24] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Analysis of a distributed e-voting system architecture against quality of service requirements. In Herwig Mannaert, Tadashi Ohta, Cosmin Dini, and Robert Pellerin, editors, *The Third International Conference on Software Engineering Advances (ICSEA 2008)*, pages 58–64, Sliema, Malta, October 2008. IEEE Computer Society.

[25] Ronald L. Rivest and Warren D. Smith. Three voting protocols: ThreeBallot, VAV, and Twin. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, August 2007. USENIX Association.

[26] André Zúquete and Filipe Almeida. Verifiable anonymous vote submission. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2159–2166, New York, NY, USA, 2008. ACM.

[27] Zhe Xia, Steve A. Schneider, James Heather, and Jacques Traoré. Analysis, improvement, and simplification of prêt à voter with paillier encryption. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, July 2008. USENIX Association.

[28] Yu-Yi Chen, Jinn ke Jan, and Chin-Ling Chen. The design of a secure anonymous internet voting system. *Computers & Security*, 23(4):330–337, 2004.

[29] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Formal verification of tamper-evident storage for e-voting. In Mike Hinchey and Tiziana Margaria, editors, *Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2007)*, pages 329–338, London, England, UK, 2007. IEEE Computer Society.

[30] J. Paul Gibson. E-voting and the need for rigorous software engineering — the past, present and future. In Jacques Julliand and Olga Kouchnarenko, editors, *B 2007: Formal Specification and Development in B, 7th International Conference of B Users*, volume 4355 of *Lecture Notes in Computer Science*, page 1, Besançon, France, 2007. Springer.

[31] J. Paul Gibson. Towards a feature interaction algebra. In Kristofer Kimbler and Wiet Bouma, editors, *FIW*, pages 217–231. IOS Press, 1998.

[32] Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer, and Dan S. Wallach. Hack-a-vote: Security issues with electronic voting systems. *IEEE Security & Privacy*, 2(1):32–37, 2004.

[33] Klaus Schmid and Holger Eichelberger. A requirements-based taxonomy of software product line evolution. *Electronic Communications of the EASST*, 8:2–13, 2008. Software Evolution 2007.