Modelling an e-voting domain for the formal development of a Software Product Line: when the implicit should be made explicit

J Paul Gibson and Jean-Luc Raffy

Abstract There has been much recent interest in the development of electronic voting (e-voting) systems, but there remain many outstanding research challenges for software and system engineers. Software product line (SPL) techniques offer many advantages for the practical development of reliable and trustworthy e-voting systems, but the composition of system features poses significant problems that can be addressed satisfactorily only through the use of formal methods. When such systems are used in government elections then they are obliged to follow legal standards and/or recommendations written in natural language. For the formal development of e-voting systems it is necessary to build a domain model which is consistent with the legal requirements. We have already demonstrated that Event-B models can be used to verify critical requirements for e-voting system components. However, the refinement-based approach needs to be applied to the engineering of a complete e-voting system. We report on our approach, using Event-B contexts to model an e-voting ontology, and its integration with an e-voting features model tree which formally specifies the SPL. During this work, we identified the importance of making the implicit explicit in two different ways — domain experts need to explicitly model implicit knowledge, and Event-B modellers need to explicitly communicate the semantics of the formal model constructs to the domain experts. If either of these tasks is not adequately carried out then this compromises validation of the requirements model (instance of the SPL).

J Paul Gibson
SAMOVAR UMR 5157, Télécom Sud Paris , 9 rue Charles Fourier, 91011 Evry cedex, FRANCE,
e-mail: paul.gibson@telecom-sudparis.eu

Jean-Luc Raffy
Télécom Sud Paris e-mail: jean-luc.raffy@telecom-sudparis.eu

1 Introduction

Electronic voting systems are those which depend on some electronic technology — including both software and hardware — for them to function as required [13]. Initial adoption of electronic voting systems focused on *direct recording electronic* (DRE) machines, where the voting is under supervision in a controlled environment, commonly known as a voting station. These systems drew much criticism, and many problems were reported around the world [35]. Much progress has been made and there is general agreement that it is possible to develop DRE that provides a reliable, trustworthy and secure e-voting system [13]. However, there are continued reports of problems with such systems [26], which are due to poor engineering practices and lack of understanding of the complex interaction between e-voting system requirements [32]. The state-of-the-art in such machines now promotes the use of a form of voter verified printed audit trail (VVPAT) [47], and a risk-limiting audit or manual recount [30].

E-voting systems make up a family of products which share common functionality, but where each system has its own unique requirements [18]. As a consequence, they are well-suited to development of a software product line [14]. Such techniques offer many advantages for the practical development of reliable and trustworthy e-voting systems, but the composition of system features poses significant problems that can be addressed satisfactorily only through the use of formal methods. A formal domain model for e-voting systems is a critical part of any such formal development [15] and this correspond to the standard notion of ontology [11].

Previous work has demonstrated that Event-B models can be used to verify critical requirements for e-voting system components [5, 6]. Recently, it has been shown how the the refinement-based approach can be applied to the engineering of a complete e-voting system [12]. The next step in our research is the integration of the SPL approach for formal specification of system requirements, and the refinement of such a specification to a concrete implementation. We report on our ongoing work, using Event-B contexts to model an e-voting ontology, and its integration with an e-voting features model tree which formally specifies the SPL.

Our research has identified the key issue of making explicit that which may be implicit — domain experts need to explicitly model implicit knowledge, and Event-B modellers need to explicitly communicate the semantics of the formal model constructs to the domain experts. If either of these tasks is not adequately carried out then this compromises validation of the requirements model (instance of the SPL).

In general, “explicit” means clearly expressed or readily observable whilst “implicit” means implied or expressed indirectly. However, there is some inconsistency regarding the precise meaning of these adjectives [1]:

- **Logic and belief models** [29] — “a sentence is explicitly believed when it is actively held to be true by an agent and implicitly believed when it follows from what is believed.”
- **Semantic web** [42] — “semantics can be implicit, existing only in the minds of the humans [. . .]. They can also be explicit and informal, or they can be formal.”

- **Requirements engineering community** [45] — use the terms to distinguish between declarative (descriptive) and operational (prescriptive) requirements, where they acknowledge the need for "a formal method for generating explicit, declarative, type-level requirements from operational, instance-level scenarios in which such requirements are implicit".

We propose a more formal (explicit) treatment of the adjectives implicit and explicit when engineering electronic systems.

The remainder of the paper is structured as follows. Section 2 provides an overview of e-voting machines: the complexity of the requirements, the different types of implementations, and the legal aspects. Section 3 is concerned with building a formal domain model as a type of ontology, and examines the question of when the explicit shoud be implicit (if ever). Section 4 reviews some real-world examples of e-voting systems where problems have arisen because of the implicit-explicit duality. Section 5 reports on our ongoing research and development of a SPL for e-voting: using Event-B to specify a domain ontology, a SPL feature tree and a generic system architecture, in order to support a feature-driven refinement process towards a correct-by-construction implementation. The paper concludes in Section 6.

2 E-voting machines

2.1 Complex, interacting requirements

Since the earliest analysis of e-voting systems [20], it has been argued that there are many complex interactions between the different requirements that these systems may be required to meet. Much of the current research in this area is concerned with better understanding these interactions, designing and implementing systems that meet certain combinations of requirements and evaluating the use of such systems during elections:

1. Authentication [10] - how to guarantee that the person who wishes to record a vote is the person they claim to be?
2. Anonymity/Privacy/Secrecy [7] - I should be able to vote without anyone knowing how I have voted.
3. Verifiability/Auditability [49] - I should be able to check that the voting process was executed correctly.
4. Accuracy [9] - whether the votes were tabulated/counted following the election rules.
5. Usability [3] - if the voting interface is easy to use for the voters and election administrators.
6. Understandibility/Trustability [39] - Can voters understand how the system works, and can they trust that their understanding is correct?

7. Fault Tolerance/Security from attack [37] - is the system secure against attacks and tolerant of faults in its component parts (electronic or otherwise).
8. Availability [28] - can access to the e-voting system be attacked, leading to a denial of service?
9. Maintainability [18] - as requirements change can the system evolve in order to maintain correct behaviour?
10. Cost/Lifetime [35] - will the machines cost more/less than the traditional systems (over the lifetime of the system), including development and maintenance costs?
11. Openness/transparency [38] - is difficult to guarantee when parts of the system are outsourced.

There are numerous documented complex interactions between these different requirements. For example, it is difficult to provide a usable interface when applying complex cryptographic protocols for security. Verifiability and anonymity appear to be inconsistent - how can a voter demonstrate that their vote was counted incorrectly if they cannot demonstrate how they have voted? Authentication and anonymity are difficult to guarantee when a voter has to identify themselves to the same machine that will be used to record their vote. Understandibility is compromised when complex cryptographic protocols are used to provide verifiability. Fault-tolerance and auditability both increase the cost of system development and maintenance. Making the system code open source may conflict with other security requirements.

2.2 Remote Electronic Voting

Remote electronic voting (REV) permits the voter to record a vote without having to be physically present in a supervised environment [27]. The voter must use unsupervised mechanisms for recording and transmitting their vote [19] In the modern world, this will most likely be an electronic computer/device that is connected to the internet [24]. Coercion is the biggest risk [8], and authentication is a major challenge [34]. Denial of service attacks have already been observed [41]. Computer viruses and malware provide powerful attack mechanisms, that have already been developed [22].

2.3 End-to-end verifiable systems

With End-to-end verifiable systems (E2E-V) [25], voters have an opportunity to verify that their vote is cast as they intended and correctly recorded (individual verifiability) Anyone can verify that all recorded votes were properly included in the tally (universal verifiability). Such systems provide a high degree of evidence that the outcome is correct, assuming that the voters correctly performed the verifications. E2E-V systems typically use sophisticated cryptographic techniques for providing privacy (although this is not a requirement). Such protocols should guarantee that

voters do not need to blindly trust any component of the system; all components can be scrutinised so that their computation can be verified if their trustworthiness is in doubt. However, even requiring the use of E2E-V systems does not guarantee that the system will meet all the requirements.

2.4 Laws Standards and Recommendations

We must ask whether a given e-voting system is lawful, as a system which does not comply with international law should not be used in democratic elections. The fundamental principles of elections are firmly stated in: *article 25 of the International Covenant on Civil and Political Rights and article 21 of the Universal Declaration of Human Rights*. Voting systems are normally required to comply with laws at other levels of governance, for example: constitutional, national, state, regional, etc. These laws often make reference to international and national standards that must be followed. An e-voting system has a myriad of inter-related legal requirements to meet; these multiple layers do not provide solid foundations upon which to build a system — none of the layers are fixed and the texts are open to different interpretations. In some cases, there is no consistent interpretation of system requirements. When problems arise with a particular e-voting system it is for judges to decide if these were due to some aspect which could be said to be illegal. The final problem to consider is that 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.

There are four main actors in the specification and use of e-voting system requirements:

1. The standards bodies establish the requirements that all e-voting systems (within a certain geopolitical space) must meet.
2. The procurement offices establish the requirements that a specific machine must meet in order for it to be purchased for use in a specific election.
3. The manufacturers develop machines that meet the generic requirements specified by the standards bodies and the specific requirements stipulated by procurement offices.
4. The Independent Testing Authorities test the delivered machines to ensure that they meet the requirements.

Unfortunately, there is evidence that the communication and co-ordination between these actors is poorly managed [33].

3 Need for a formal domain model: ontology

3.1 Terminology: dictionaries and glossaries

The European Council of Europe e-voting recommendations [46] recognises that consistent use of terminology is key, and states: *'In this recommendation the following terms are used with the following meanings: . . . "* The terms that it chooses to include are: *authentication, ballot, candidate, casting of the vote, e-election or e- referendum, electronic ballot box, e-voting, remote e-voting, sealing, vote, voter, voting channel, voting options and voters registrar*. The list is very incomplete, but it is a first importat step towards developing an e-voting domain ontology. It should be noted that, even in this short set of definitions, fundamental terms are used inconsistently. A formal ontological model would facilitate automated validation of model consistency.

A number of other countries, outside of Europe, have also developed glossaries for their particular voting systems and requirements, for example: Canada[1], Australia [2] and USA[3]. The level of detail in such glossaries varies from a short list of terms to hundreds of pages. There is a clear need for a standard ontological domain model.

3.2 When the explicit should be implicit: ontologies and domain-specific languages

In all forms of communication, implicit shared understanding improves signal rate, and is often necessary in achieving an acceptable communication mechanism. Shared implicit understanding is good when it is coherent. There should always be an explicit representation of the implicit knowledge as a base reference, if needed. This is the role of an ontology [21].

When a community of developers share much common knowledge then the next step is the development of a domain-specific language (DSL) [44]. Many such DSLs are structured in terms of domain features [43], and this provides a strong link to the SPL modelling and development approach. Building a DSL is a complex task [36], and there has been much recent research on using formal approaches [4]. Integration of DSLs and ontologies requires the use of formal methods [48].

With respect to the implicit/explcit dichotomoy, such DSLs bring the best of both worlds. Shared domain knowledge is implicit when the DSL is used to describe and synthesise systems within the domain, but is also explicit when used to reason about and analyse such systems. The implicit aids human-human interaction and commu-

[1] `www.elections.ca`

[2] `www.aec.gov.aufooterGlossary.htm`

[3] `www.eac.govvoting-equipmentvoluntary-voting-system-guidelines`

nication. The explicit aids automation and tool development. Both are necessary for model validation.

4 E-voting: Examples of when the implicit should be explicit

4.1 DUALVOTE - e-pen

The authors have been involved in the development of a novel e-voting system (DUALVOTE) that provides an innovative interface for e-voting using an electronic pen [31]. The advantages of the system arise out of the way in which it generates a paper vote (for audit) and an electronic vote simultaneously. However, there were some problems that arose when testing the system (during real elections) because of the unpredictable behaviour of the voters [17].

A major issue was that the developers had made implicit asssumptions about voter behaviour which had never been explicitly stated to the voters. The correct functionality of the system was dependent on these assumptions being true. Unfortunately, this was not the case:

- Some voters recorded their vote using their own pen rather the e-pen that was provided. We wrongly assumed that all voters would record their votes using the e-pen.
- Some voters recorded their vote on a surface other than the electronic surface provided. The instructions explicitly stated that they should write on the surface provided, but it was wrongly assumed that all voters would follow the instructions.
- It was explicitly stated that any identifying mark on the ballot paper which could uniquely identify the voter would render the vote invalid. However, there was no explicit statement of how a machine could automated the identification of such invalid votes. The system failed to function correctly because the election administrators had implicit (domain specific) knowledge of how voters could mark their vote which had never been explicitly stated. As an additional complication, this knowledge varied from election-to-election and from constituency-to-constituency. There were even disagreements between election officials in the same voting station as to what would render a vote invalid.
- The electronic voting interface was a limited resource and tying up the resource could lead to a denial of service type attack. We implicitly, and wrongly, assumed that all voters would spend a reasonable amount of time to vote. This was based on us wrongly assuming that voters would not deliberately attack the voting system. A simple attack, in this case, would be to stay at the voting interface for a long (unreasonable) amout of time. The implicit need to timeout a voter should have been explicitly stated and defined before the voting process started.

Our DUALVOTE experience highlighted the need to make the implicit explicit when developing voting systems.

4.2 Implicit Programming Language Semantics

While analysing an e-voting system we identified issues that arise when system components are developed independently, and they make inconsistent assumptions about the global system and its environment. A good example of this arises when system software is written using different languages. The meaning of the software behaviour is now dependent on the semantics of the programming language concepts. Unfortunately, concepts that share the same syntax (in different languages) do not always share the sme semantcs.

This can happen with concepts as simple as arrays [16]. In this study, two components of the system were developed using different modelling/programming languages/techniques. Each of the developers had correct implicit understanding of the semantics of arrays in the language they were using (and an explicit statement of these semantics was available). However, the semantics of arrays was different in each of the two languages. As votes moved from being recorded in one language to being counted in another language, their representations changed. The inconsistency was caught late in development (during testing), but it would be better if such issues had been avoided at the beginning of development.

4.3 Negative counts - can anything be too obvious?

It is obvious that a vote count for a specific option (or candidate) should not be negative. Returning a negative count is clearly an error. Such negative counts have been reported in real elections. We must ask how such a stupid error could have happened. Should non-negativity of counts be explicit in the requirements specification? In order to answer this question we need to examine a major difference between modelling languages and programming languages with respect to integer representation.

In Event-B, there are 3 in-built types that can be used for counting: integers, naturals (including 0) and naturals (excluding 0). If we model a count as a natural (`NAT`) then the model guarantees that the count has a non-negative value. There should be no need to explicitly state this as it is implied by the semantics of `NAT`. However, if the client (the person specifying the system requirements) does not know/understand the formal semantics of Event-B then how can they validate the model as being correct? After validation, the model has to be implemented. Most programming languages do not contain the notion of a natural number as a primitive type. As a consequence, it is not surprising that a count could be implemented as an integer (which does permit negative values). In such a case, a program invariant (stating that the count is always non-negative) would have solved the problem. This property of `NAT`s is implicit in the Event-B model; thus the the non-negative count invariant is not explicitly stated and may be overlooked in an integer-based implementation. Of course, using a refinement-based development method will prohibit an incorrect implementation.

A similar issue occurs with many other model language semantics. A good example is that of sets. The modeller knows that a collection of entities modelled as a set implicitly guarantees that the collection contains no repeated elements; but the client may not know this. In e-voting, it is often very important that collections have such set-like behaviour. How should we validate such models?

4.4 Vote Coercion in a Typical Voting Station

Formal methods are generally used in the development of electronic systems. However, they can also be used in modelling and analysing traditional voting systems (with no electronic components). Insight from the formal development of e-voting systems has helped us to identify previously undocumented issues with the traditional systems. Without a formal model — explicitly modelling assumptions — the correct functioning of a traditional system can be guaranteed only if the implicit assumptions are valid. Unfortunately, many of these implicit assumptions have not been explicitly documented and are true only because the actors in the system behave in a certain way. A simple example from voting a polling stations in France illustrates this case.

In remote voting, there is a major threat of man-in-the-middle attacks on the communication network between the voting booth and the ballot box. Such attacks have been modelled using formal methods and we have a good understanding of how systems can be defended against such attacks. By explicitly modelling such attacks, we can verify that our systems are protected against them. It has been claimed that no such threats exist when using traditional paper voting at a controlled voting station, as there is no underlying network to be attacked. Such reasoning is based upon a false assumption that is implicit in the correct functioning of a traditional voting station; namely, that voters do not interact with other parties between recording their vote anonymously in the booth and the submission of the recorded vote in the ballot box . In France, and in other countries, the passage between the booth and the box is not strictly controlled. The implicit need for voters to transfer their vote directly from the booth to the box is, in general, not explicitly enforced. This could lead to problems of coercion and vote buying. The problem is mitigated by the fact that the polling station is a public area where voters can be observed; but there is no guarantee that invalid behaviour of voters will be witnessed.

5 When the explicit should be implicit: ontologies and domain-specific languages

In all forms of communication, implicit shared understanding improves signal rate, and is often necessary in achieving an acceptable communication mechanism. Shared implicit understanding is good when it is coherent. There should always be

an explicit representation of the implicit knowledge as a base reference, if needed. This is the role of an ontology [21].

When a community of developers share much common knowledge then the next step is the development of a domain-specific language (DSL) [44]. Many such DSLs are structured in terms of domain features [43], and this provides a strong link to the SPL modelling and development approach. Building a DSL is a complex task [36], and there has been much recent research on using formal approaches [4]. Integration of DSLs and ontologies requires the use of formal methods [48].

With respect to the implicit/explcit dichotomoy, such DSLs bring the best of both worlds. Shared domain knowledge is implicit when the DSL is used to describe and synthesise systems within the domain, but is also explicit when used to reason about and analyse such systems. The implicit aids human-human interaction and communication. The explicit aids automation and tool development. Both are necessary for model validation.

6 A SPL for e-voting

In this section we provide a brief overview of our current work in developing a formal SPL for e-voting.

6.1 A feature tree model for e-voting

In Fig. 1 we illustrate part of the feature tree for an early version of our SPL prototype.

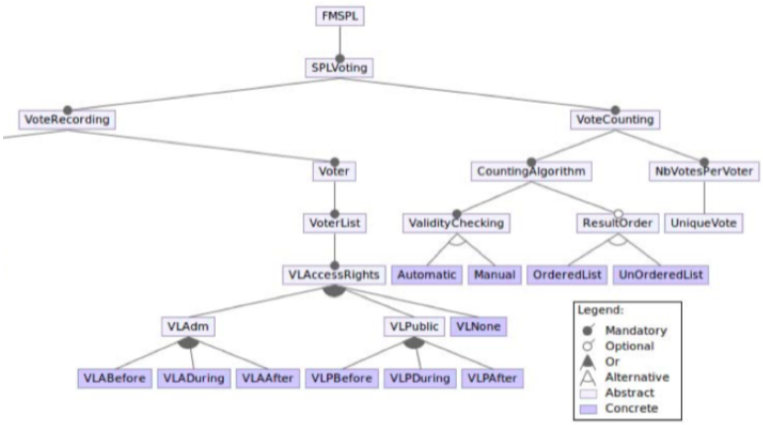

Fig. 1 The (partial) feature tree for an e-voting SPL

From the diagram (which shows only part of the full feature tree model) we see that `VoteCounting` is a mandatory abstract feature of our e-voting SPL. `ValidityChecking` is a mandatory (sub) feature of `VoteCounting` which can be done either automatically or manually (which are modelled as concrete features). In the `VoteRecording` branch, voter list access rights (`VLAccessRights`) must be defined, and may be given to the administrators and/or the public. The details of individual requirements associated to each feature are not important. We note that the number of features (in the complete tree) was chosen to provide a SPL which was simple enough to develop, but complex enough to provide a proof-of-concept. In our current version, our SPL can be configured into hundreds of different concrete instances.

6.2 Formalisation in Event-B

The tree is specified formally in an Event-B generic context, which is generated automatically from a feature-tree graphical editor. The configuration of the specific instance of the SPL can also be done interactively using a graphical editor. Again, we generate automatically the Event-B context corresponding to the instance. At this stage, the formal methods guarantee that the tree instantiation is a valid instance of the SPL, respecting the constraints specified in the feature tree model. We now have to associate behaviour with each of the features in the tree. Currently, we follow a 3-tier approach:

1. Each feature has an associated context where static relationships between sets and constants are specified. These sets and constants are taken from an e-voting domain ontology context. (Fig. 2 illustrates the context that is generated for an instance corresponding to the second round of a presidential election in France.)
2. Each feature has an associated state, which is a union of all system variables that it is concerned with. Features specify an invariant over the relevant state. The state variables are also part of the domain ontology
3. Each feature can be associated to one or more events that correspond to changes to its relevant state variables. The events are also modelled in the domain ontology.

As such, the domain ontology groups together all the concepts (static and dynamic) that are shared between the SPL features. More details of the formalisation of the feature tree can be found in [2].

Combining these 3-tiers into a single abstract Event-B machine is the main challenge that we are currently addressing.

CONTEXT PresidentialRound2
SETS
BULLETINS
REGISTEREDVOTERS
CONSTANTS
BLANKVOTE
NUMBEROFOPTIONS
CANDIDATES
INVALIDVOTE
POSSIBLEOPTIONCHOICE
MAXNUMBEROFPREFERENCES
MINNUMBEROFPREFERENCES
AXIOMS
axm1: $BLANKVOTE \in BULLETINS$
axm2: $NUMBEROFOPTIONS \in \mathbb{N}$
axm3: $CANDIDATES \subseteq REGISTEREDVOTERS$
axm4: $INVALIDVOTE \in BULLETINS$
axm5: $POSSIBLEOPTIONCHOICE = 1$
axm6: $NUMBEROFOPTIONS = 2$
axm7: $MAXNUMBEROFPREFERENCES = 1$
axm8: $MINNUMBEROFPREFERENCES = 1$
axm9: ⟨theorem⟩ $MAXNUMBEROFPREFERENCES \leq NUMBEROFOPTIONS$
axm10: ⟨theorem⟩
$MINNUMBEROFPREFERENCES \leq MAXNUMBEROFPREFERENCES \wedge$
$MINNUMBEROFPREFERENCES > 0$
END

Fig. 2 The instantiated context for Presidential Election Round 2

6.3 A pipeline design pattern

The initial abstract machine is modelled as a simple pipeline, of 4 phases - set up, voting, counting and audit (see Fig. 3). We have identified a useful formal design pattern, where a state variable in one phase becomes a constant in the next phase:

- During election set up the list of candidates and list of electors is variable (as they must register before being added to the lists). However, once set up is completed these lists must be fixed.
- During the voting step, the list of electors who have voted is variable (names get added as votes are recorded). However, once the polling station closes this list is fixed before the count begins.
- Counting, in phase 3, will be complete when the list of ballots counted is complete,, and then we can start the final audit phase.

We note that the precise semantics of the pipeline operator have recently been published [12]. We also note that the same paper addresses a correct-by-construction refinement approach using the same architecture. However, the initial abstract machine in this work is not initially generated from the SPL feature tree. The development of the feature tree model and the validation of the pipe-line approach were done in parallel, and, as such, are not yet fully integrated.

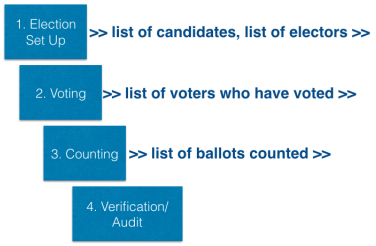

Fig. 3 The pipeline architecture

6.4 Feature-driven refinement towards implementation

Developing features as a sequence of refinements guarantees correctness if such a sequence can be found. The order in which features/refinements are added is very important. We propose that such refinement should be driven by the feature configuration. Developing features in parallel poses many additional issues with respect to feature interactions [14]. We can usefully classify feature pairs in terms of the way in which they interact following the approach for composing fair objects [23]. As we identify (enemy) features that have contradictory requirements, we update the feature tree with a constraint to preclude their composition. As we identify (politician) features that need special co-ordination in order for them to work correctly, we refine the system in such a way that guarantees such a co-ordination. Features that require no special co-ordination in order to work correctly together are known as friends and their refinements can be safely developed in parallel (unlike with politicians). Development of a feature-driven refinement method is a major challenge in the formal development of software product lines [**?**], and is the major challenge in our future research..

7 Conclusions

We have presented our on-going research and development of a formal SPL for e-voting. We have placed this work within the context of domain ontologies and demonstrated the need to better model and understand the implicit-explicit semantic dichotomy. Even if we trust the systems that we develop formally using our approach, we still do not know how we can get the public to trust them. We argue that if the SPL is trustworthy then it guranatees the trustworthiness of every machine that is built using it. Using our approach does not guarantee the absence of unwanted feature interactions. However, it will aid in detecting such interactions and make explicit to the client the incompatibilty between certain features. In the future, we would like to develop a prescriptive assurance case mechanism (such as seen in [40]) based around the SPL model. As we configure and develop more and

more e-voting systems using our approach, we may consider the SPL e-voting feature tree as a type of Domain Specific Language which evolves as we improve our understanding of feature interactions. As future work, we are interested in whether the formal SPL domain language ontological approach generalise to other problem domains.

References

1. Yamine Ait-Ameur, J.Paul Gibson, and Dominique Méry. On implicit and explicit semantics: Integration issues in proof-based development of systems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, volume 8803 of *Lecture Notes in Computer Science*, pages 604–618. Springer Berlin Heidelberg, 2014.
2. Abderrahim Ait Wakrime, J Paul Gibson, and Jean-Luc Raffy. Formalising the Requirements of an E-Voting Software Product Line using Event-B. In *27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 78–84, Paris, France, June 2018. IEEE.
3. Benjamin B Bederson, Bongshin Lee, Robert M Sherman, Paul S Herrnson, and Richard G Niemi. Electronic voting system usability issues. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–152. ACM, 2003.
4. Jean-Paul Bodeveix, Mamoun Filali, Julia Lawall, and Gilles Muller. Formal methods meet domain specific languages. In *Integrated Formal Methods*, pages 187–206. Springer, 2005.
5. Dominique Cansell, J. Paul Gibson, and Dominique Méry. Formal verification of tamper-evident storage for e-voting. In *SEFM*, pages 329–338. IEEE Computer Society, 2007.
6. 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.
7. Chin-Ling Chen, Yu-Yi Chen, Jinn-Ke Jan, and Chih-Cheng Chen. A secure anonymous e-voting system based on discrete logarithm problem. *Applied Mathematics & Information Sciences*, 8(5):2571, 2014.
8. Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *Financial Cryptography*, volume 7035, pages 47–61. Springer, 2011.
9. Dermot Cochran and Joseph R Kiniry. Formal model-based validation for tally systems. In *International Conference on E-Voting and Identity*, pages 41–60. Springer, 2013.
10. Stefanie Falkner, Peter Kieseberg, Dimitris E Simos, Christina Traxler, and Edgar Weippl. E-voting authentication with qr-codes. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 149–159. Springer, 2014.
11. Dieter Fensel. *Ontologies*. Springer, 2001.
12. J. Paul Gibson, Souad Kherroubi, and Dominique Méry. Applying a dependency mechanism for voting protocol models using Event-B. In Ahmed Bouajjani and Alexandra Silva, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, volume 10321 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2017.
13. J. Paul Gibson, Robert Krimmer, Vanessa Teague, and Julia Pomares. A review of e-voting: the past, present and future. *Annals of Telecommunications*, 71(7):279–286, 2016.
14. J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Feature interactions in a software product line for e-voting. In Nakamura and Reiff-Marganiec, editors, *Feature Interactions in Software and Communication Systems X*, pages 91–106, Lisbon, Portugal, June 2009. IOS Press.

15. J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Engineering a distributed e-voting system architecture: Meeting critical requirements. In Holger Giese, editor, *Architecting Critical Systems, First International Symposium, ISARCS 2010, Prague, Czech Republic, June 23-25, 2010, Proceedings*, volume 6150 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2010.
16. J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Formal object oriented development of a voting system test oracle. *Innovations in Systems and Software Engineering (Special issue UML-FM11)*, 7(4):237–245, September 2011.
17. J. Paul Gibson, Damien MacNamara, and Ken Oakley. Just like paper and the 3-colour protocol: a voting interface requirements engineering case study. In *Proceedings of 2011 International Workshop on Requirements Engineering for Electronic RE-Vote 2011*, pages 66–75, Trento, Italy, august 2011. IEEE.
18. 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, Lausanne, Switzerland, July 2008. Academic Publishing International.
19. Gurchetan S Grewal, Mark D Ryan, Liqun Chen, and Michael R Clarkson. Du-vote: Remote electronic voting with untrusted computers. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 155–169. IEEE, 2015.
20. Dimitris Gritzalis. Principles and requirements for a secure e-voting system. *Computers & Security*, 21(6):539–556, 2002.
21. Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, 43(5-6):907–928, 1995.
22. J Alex Halderman. Practical attacks on real-world e-voting. *Real-World Electronic Voting: Design, Analysis and Deployment*, pages 145–171, 2016.
23. Geoff Hamilton, J. Paul Gibson, and Dominique Méry. Composing fair objects. In Fouchal and Lee, editors, *International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD '00)*, pages 225–233, Reims, France, May 2000.
24. David Jefferson, Aviel D Rubin, Barbara Simons, and David Wagner. Analyzing internet voting security. *Communications of the ACM*, 47(10):59–64, 2004.
25. Rui Joaquim, Paulo Ferreira, and Carlos Ribeiro. Eviv: An end-to-end verifiable internet voting system. *computers & security*, 32:170–191, 2013.
26. Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy (S&P04)*, pages 27–40. IEEE, 2004.
27. Robert Krimmer and Melanie Volkamer. Bits or paper? comparing remote electronic voting to postal voting. In *EGOV (Workshops and Posters)*, pages 225–232, 2005.
28. Thomas W Lauer. The risk of e-voting. *Electronic Journal of E-government*, 2(3):177–186, 2004.
29. Hector J. Levesque. A logic of implicit and explicit belief. In Ronald J. Brachman, editor, *AAAI*, pages 198–202. AAAI Press, 1984.
30. Mark Lindeman and Philip B Stark. A gentle introduction to risk-limiting audits. *IEEE Security & Privacy*, (5):42–49, 2012.
31. Damien MacNamara, J. Paul Gibson, and Ken Oakley. A preliminary study on a dualvote and prłt voter hybrid system. In *International Conference for E-Democracy and Open Government 2012*, pages 77–89, Danube University Krems, Austria, May 2012. Edition Donau-Universitt Krems.
32. 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.
33. 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 2006 on Electronic Voting Technology Workshop*, pages 1–13, Berkeley, CA, USA, 2006. USENIX Association.
34. Bo Meng. A secure non-interactive deniable authentication protocol with strong deniability based on discrete logarithm problem and its application on internet voting protocol. *Information Technology Journal*, 8(3):302–309, 2009.

35. Rebecca Mercuri. A better ballot box? *IEEE Spectr.*, 39(10):46–50, 2002.
36. Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
37. Peter G Neumann. Security criteria for electronic voting. In *16th National Computer Security Conference*, volume 29, 1993.
38. Anne-Marie Oostveen. Outsourcing democracy: losing control of e-voting in the netherlands. *Policy & Internet*, 2(4):201–220, 2010.
39. Julia Pomares, Ines Levin, R Michael Alvarez, Guillermo Lopez Mirau, and Teresa Ovejero. From piloting to roll-out: voting experience and trust in the first full e-election in argentina. In *Electronic Voting: Verifying the Vote (EVOTE), 2014 6th International Conference on*, pages 1–10. IEEE, 2014.
40. Thomas Rhodes, Frederick Boland, Elizabeth Fong, and Michael Kass. Software assurance using structured assurance case models. *Journal of research of the National Institute of Standards and Technology*, 115(3):209, 2010.
41. Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J Alex Halderman. Security analysis of the estonian internet voting system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–715. ACM, 2014.
42. Michael Uschold. Where are the semantics in the semantic web? *AI Mag.*, 24:25–36, September 2003.
43. Arie Van Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *CIT. Journal of computing and information technology*, 10(1):1–17, 2002.
44. Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.
45. Axel van Lamsweerde and Laurent Willemet. Inferring declarative requirements specifications from operational scenarios. *IEEE Trans. Softw. Eng.*, 24:1089–1114, December 1998.
46. Venice Commission. Code of good practice in electoral matters. CDL-AD (2002) 23, 2002.
47. Adolfo Villafiorita, Komminist Weldemariam, and Roberto Tiella. Development, formal verification, and evaluation of an e-voting system with vvpat. *Trans. Info. For. Sec.*, 4(4):651–661, 2009.
48. Tobias Walter and Jürgen Ebert. Combining dsls and ontologies using metamodel integration. In *DSL*, pages 148–169. Springer, 2009.
49. Xukai Zou, Huian Li, Feng Li, Wei Peng, and Yan Sui. Transparent, auditable, and stepwise verifiable online e-voting enabling an open and fair election. *Cryptography*, 1(2):13, 2017.