Applying a Dependency Mechanism for Voting Protocol Models Using Event-B

Paul J. Gibson, Souad Kherroubi, Dominique Méry

Applying a Dependency Mechanism for Voting Protocol Models Using Event-B*

J. Paul Gibson[1] Souad Kherroubi[2] and Dominique Méry[2]

[1] Telecom Sud Paris, SAMOVAR UMR 5157 CNRS Research Laboratory, METHODES Team, France.
[2] Université de Lorraine , LORIA UMR 7503 CNRS Research Laboratory, MOSEL Team, France.

Abstract. The design of e-voting systems requires the use of techniques which guarantee that the resulting system is safe, secure and preserves privacy. We develop Event-B models of a voting system, by applying a decomposition pattern and a technique of contextualisation, using a *dependency mechanism*. Through refinement, we take into account the precise regulation and structure of a specific voting process, and reason formally about the system's resistence to common attacks and threats.

1 Introduction

In general, elections are critical processes concerned with the collection, recording and counting of votes [9]. All election processes use protocols satisfying security, safety and privacy properties, which are difficult to express and to validate. We have applied a correct-by-construction refinement technique to formally model and reason about a voting process. The formal approach helps us to validate the coherency of different types of interacting assumptions and requirements [10].

1.1 Diffferent Points of View

There are many different points of view concerning elections. Firstly, citizens are mostly not overly-concerned with the interacting tasks used in reaching the decision. They refer to abstract processes such as *voting* and *counting*, without fully understanding the subtle details. Secondly, e-voting domain experts are concerened with the complexity of modelling the election process at different levels of abstraction. From our, third, point of view, as system engineers, a voting process is managed by a system which facilitates voting, whilst satisfying the requirements of the vote with respect to the current legal position. When the system is electronic, it may also have to meet legal requirements which are not relevant for the traditional manual systems. A final point of view is one of security: voting systems make use of information and communication technologies (ICT), and their dependability relies on security analysis for identification of threats in order to select countermeasures.

* This work was supported by grant ANR-13-INSE-0001 (The IMPEX Project `http://impex.gforge.inria.fr` (or `http://impex..loria.fr`) from the Agence Nationale de la Recherche (ANR).

1.2 Contextual Reasoning

We [12] have previously shown the importance of context in proofs, where it captures the system designer's intention, as well as giving the system model a precise and unambiguous semantics. Our study demonstrates that context is always related to an activity, a focus or a situation. More precisely, the context is a *"moment universals"* that depends on an intentional concept i.e. "action". By reasoning over the structure of the Event-B, the *context of proof* is decomposed into — *i) Constraints*: conditions having their own existence and concerned with the theory defined for Event-B, corresponding to the sets, constants and axioms defined in the Event-B contexts; *ii)Hypotheses*: that are assumed to be true, but not always verified, and which are expressed by restrictions on the constraints, and suppositions on the corrupt behaviours in the system. *iii)Dependencies*: this knowledge is deduced, and expressed as a combination of situations and constraints over time. The use of dependencies was inspired by the work of [17,2,8], and led us to formalize a dependency mechanism in Event-B as a proof of the coherency of the contexts in Event-B.

1.3 Refinement and Decomposition Patterns

The correct-by-construction approach [16] can be applied for integrating progressively *properties and details of the voting process*. In the case of the voting system, we decompose it into three dependent and sequential phases: the *preparation* phase, the *recording* phase and the *tallying* phase. These phases are sequential and linked in a pipeline, where the activation of the next phase depends on the termination of the previous one. One phase may use data computed during the previous phase; this data is dynamically generated in one phase but is then used to statically instantiate the configuration parameters of the next phase. We have defined this approach as a domain-independent re-usable template, using a formal *dependency pattern*, defined in a separate work [12]. Patterns [11] are applied to refinement-based processes; they help to increase productivity. and improve quality by providing guarantees with respect to avoidance of security risks and attacks [14]. We use the *sequential decomposition* pattern and identify the three phases characterized by three main liveness properties: (1) *preparation* collects information for defining the persons authorized to vote and candidates/options authorized to be presented as choices in the election; (2) *recording* permits authorized voters to choose their preferred candidate(s) or option(s); and (3) *tallying* counts the votes for each candidate, or given option. Thus, our three stage pipeline is a composition of two instances of the *sequential decomposition* pattern.

1.4 Formal Reasoning about E-voting

Many properties and requirements are expressed in the literature of e-voting systems. We follow the reasoning of [7] which argues that, in the ideal case, a secure

voting system should guarantee eligibility, confidentiality, anonymity and verifiability. *Verifiability* ensures that all voters can trust the proclaimed result without having to trust a particular authority or actor in the system. Furthermore, it ensures the existence of an algorithm that can exhibit the proof of the result of tallying and integrity of the authorities. In our case, the proof obligations generated show that the system has behaved correctly. *Confidentiality* guarantees knowledge of each voter is limited only to his/her vote. The *Anonymity* of a vote is guaranteed by breaking the link between the voter and theirr vote. *Eligibility* of voters determines whether or not a voter is entitled to vote. Our model expresses this property as a condition concerning the *authentication* and *authorization* of each voter before recording their vote. *Authentication* identifies voters using *credentials* and *passwords* previously provided for this purpose.

2 The Modelling Framework

Event-B is a formal language well-suited to the modelling of *reactive* systems that respond to external stimuli over time. In this set-theoretic language in first-order logic (FOL), guarded *events* provide state transition behaviour. The two syntactic units of structuring are the static *context* and the dynamic *machine*. The context comprises sets, constants, axioms, and any theorems that must be derived from those axioms. The machine comprises dynamic variables and the events that update them. Safety properties are expressed as either invariants or theorems. Every machine *sees* at least one context.

An event is observed in a model with constants c and sets s subject to *axioms* $P(s, c)$ and an *invariant* $I(s, c, v)$. Consistency proof obligations (POs) require that events are well-defined, feasible and maintain invariants. The term *refinement* is overloaded, referring both to the process of transforming models, and to the more concrete model which refines the abstract one. When model $N(w)$ refines $M(v)$, it contains a refinement relation, or "gluing invariant" $J(s, c, v, w)$. New events may be introduced in refinement to act on new variables, effectively refining stuttering steps (called "skip" in Event-B). The refinement POs enforce the standard forward simulation refinement rule [1] that every concrete step of a refining event re-establishes the gluing invariant subject to some corresponding step of the abstract refined event, or skip.

Figure 1 summarizes the two kinds of models that are used in the formal development. In this work the modelling process deals with various languages, as seen by considering the triptych of Bjoerner [5]: $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$. Here, the domain $\mathcal{D}$ deals with properties, axioms, sets, constants, functions, relations, and theories. The system model $\mathcal{S}$ expresses a model or a refinement-based chain of models of the system. Finally, $\mathcal{R}$ expresses requirements for the design of the system. One must note that the Event-B modelling language is not expressing liveness properties and we follow the methodology introduced by Méry and Poppleton [18] for managing such properties. We will use a notation from TLA to express liveness properties under fairness assumptions. We have to interpret our Event-B models over traces generated from the Event-B machines and we extend the scope of

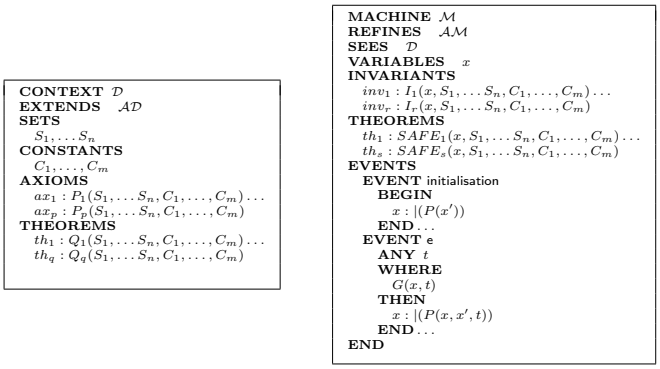

Fig. 1. Context and Machine

the Event-B machines by using TLA as follows. Let M be an EVENT B machine and D a context seen by M. Let x be the list of variables of M, let E be the set of events of M, and let $Init(x)$ be the initialisation event in M. The temporal framework of M over D is defined by the TLA specification denoted: $\mathcal{S}pec(M) \triangleq BA(Init)(x) \wedge \Box[Next]_x \wedge \text{FAIR}$, where $Next \equiv \exists e \in E.BA(e)(x, x')$ and FAIR defines the fairness assumptions.

Following Lamport [15]the specification $spec(M)$ is valid for the set of infinite traces simulating M with respect to the events of M. $\mathcal{S}pec(M)$ is thus defined by the initial conditions, the next relation and fairness constraints. In practice we have to discover the weakest fairness assumptions, denoted FAIR(M), that allow us to derive the required liveness properties. These fairness assumptions emerge from the proof rules applied, and are expressed in terms of the temporal operators of TLA, namely WF and SF . FAIR(M) is thus a combination of fairness operators over events of M. Liveness properties for M are, de facto, defined in TLA as follows: M satisfies $P \rightsquigarrow Q$, when $\Gamma(M) \vdash \mathcal{S}pec(M) \implies (P \rightsquigarrow Q)$. When deriving the proof of $\mathcal{S}pec(M) \implies (P \rightsquigarrow Q)$, we apply the right introduction rule of the implication and then we eliminate the conjunctive connective in the left part of the $\vdash$ symbol. Thus $\Gamma(M)$ will be increased by fairness assumptions and we can use an alternative form for expressing the initial sequent: $\Gamma(M)$ is the proof context of M. In review, the refinement of Event-B models preserves the safety properties; and for preserving the liveness properties we follow the technique proposed by Mery and Poppleton [18] (see Fig. 2).

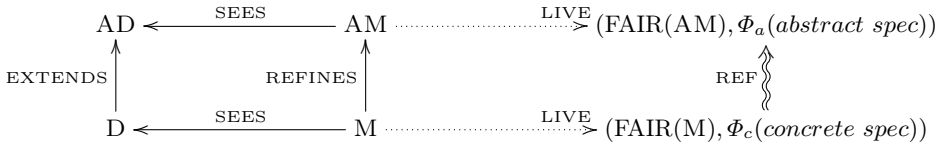

Fig. 2. Summary of the integrated formal methods refinement methodology

3 Modelling the Voting System

The 3 phases of our voting process are each developed, and verfified, in a separate refinement chain. In this paper, we present only the final 2 stages of the pipeline: the vote recording and the tally (count) phases. The system description also includes the conditions over the environment that express voter behaviour and possible attacks on the system. In particular, our development disregards different roles and/or actors in the system: the only actor represented is the voter who interacts with the system interface. In particular, our refinement-based approach takes into account intruders with the following capabilities: *1)establishing a connection*; *2)closing an already established session*; *3)making choices*; *4)adding signatures*; *5)adding ballots: ballot stuffing*; *6)adding signatures and ballots simultaneously*; *7)removing signatures*; *8)removing ballots*; and *10)accessing signatures*, *credentials*, *passwords*. These different assumptions concerning corrupt behavior correspond to the part of the world in which the system is immersed. They situate the developed system and we qualify them as ***"context of assumptions"***.

3.1 Combining Refinement and Composition, using the Dependency Pattern

Figure 3 illustrates the refinement-based approach followed in our development, and shows the use of the dependency pattern mechanism (`depends`) to compose the machines associated with sequential phases of the voting process.

3.2 Refining the Voting Phase in Seven Steps

The first phase is described by an Event-B context $C0\_Recording$ which defines the constraints and static elements that are seen by the 9 machines in our development. The first Event-B context introduces the necessary elements to start a recording phase of votes ie: sets, constants and static properties such as *Electors*, *Choices*, *Envelopes*, *PollStation*, *Representatives*, *Bulletins*, *Sig*, *electoral_roll*, *voters_hosting*, *start_time*, *end_time* etc.....

Abstract model - In this first model the state of the system is characterized by two variables that represent the registered votes and the elapsed time in the system. The votes are modelled as a relationship between all signatures (*Sig*) and the electors' choices (*Choices*). The invariant in this machine simply provides a means to type these variables. The precondition for this phase, as expressed by the initialization event, is that the time is equal to the opening time of the offices fixed in the context $C0\_Recording$ and that no vote has been recorded. A vote modifies the variable *rec_votes* which is performed by the event *register_votes*. In this model, we distinguish only the values of variables *rec_votes* which take their values in $Sig \leftrightarrow Choices$ without precising the undertaken actions. The event $forwarding\_time$ changes the value of the variable *timer* introduced in

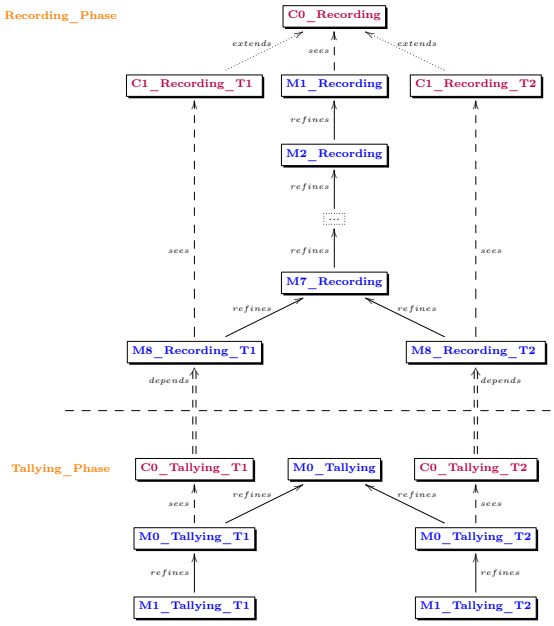

Fig. 3. Structure of the refinement-based formal development of the voting system

this machine to express the progression of time in the system. The variable value is incremented by the action of thes event $forwarding\_time$ until the closing time of the offices end_time is reached. We note that this event has a convergent status under which a weak fairness assumption is made. Thus, this event ($forwarding\_time$) will not be observable when the value of the $timer$ variable has reached end_time. Note that the vote event can be observed only when the voting has begun and that the closing time has not yet been reached (see *grd1*). The convergence of these events is proved using a simple variant. Then, at the end of voting, no one can cast a vote or record a signature: the only event that will be observable is $finish$.

VARIABLES $rec\_votes, timer$
INVARIANTS
$inv1 : rec\_votes \in Sig \leftrightarrow Choices$
$inv2 : timer \in start\_time \mathinner{..} end\_time$
VARIANT $end\_time - timer$
INITIALISATION
$act1 : rec\_votes := \varnothing$
$act2 : timer := start\_time$
EVENT *register_votes*
WHEN
$grd1 : timer \geqslant start\_time \wedge timer < end\_time$
$grd2 : \forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$
THEN
$act1 : rec\_votes : |rec\_votes' \in (Sig \leftrightarrow Choices)$
END

EVENT forwarding_time
STATUS convergent
WHEN
$grd : timer < end\_time$
THEN
$act : timer := timer + 1$
END
EVENT finish
WHEN
$grd1 : timer = end\_time$
THEN
$act : skip$
END

In the following seven refinements, termination proofs are the same as those for this initial abstract machine. Since all events that will be introduced in

the following will also be guarded by the guards *grd1* and *grd2* of the event *register_ votes*, no event changes the time and $forwarding\_time$ and $finish$ remain unchanged. We also recall that all variables introduced in the followed refinements are initialized with the empty set, with the exception of intruder knowledge variables.

Refinement 1 - distinguishes the votes that are either registred or deleted. The recording of votes is done by the event *register_ votes* refined by itself, while tdeleting votes is done by the event *remove_ votes*. Both events refine the former introduced in the abstract model *register_ votes*.

Refinement 2 - introduces intrusion scenarios, where people present themselves to vote on behalf of someone else, without having permission to do so. Such a scenario is a single example of one among many forbidden scenarios that may exist. We distinguish at this level of refinement, the votes recorded correctly and those that are corrupted. Four new variables (*valid_ sig*, *valid_ choices*, *valid_ votes*, *corrupt_ votes_ sig*) are introduced:

INVARIANTS

$inv1 : valid\_sig \subseteq Sig$
$inv2 : valid\_choices \subseteq Choices$
$inv3 : valid\_votes \in valid\_sig \rightarrowtail\!\!\!\!\!\!\!\!\rightarrow vote$
$inv4 : corrupt\_votes\_sig \in Sig \leftrightarrow Choices$
$inv5 : valid\_votes \subseteq rec\_votes$
$inv6 : corrupt\_votes\_sig \subseteq rec\_votes$
$inv7 : corrupt\_votes\_sig \cap valid\_votes = \varnothing$

The votes or choices are identified in the set of choices ($inv2$), while the signatures are a subset of the set *Sig* ($inv1$) defined in the Event-B context $C0\_Recording$.

The property $inv3$ associates each correct choice with one and only one signature, and each signature with one and only one correct choice. Thus, at any time the number of votes at the polls equals the number of signatures honestly recorded. Votes can be corrupt, but these are detected. The invariant property $inv7$ indicates that the correct votes and the corrupt votes partition the set of all votes cast. Others variables are also introduced separately in order to identify corrupt signatures and invalid choices. The event to register the votes introduced in the first model is refined into two events that allow the storage of both types of vote. This corruption scenario is one in which corrupt choices and corrupted signatures are introduced simultaneously via the event *corrupt_ choices_ sig_ simultaneously*. This refinement also introduces two other scenarios of corruption consisting of stuffing ballot boxes, or recording votes, without valid signatures. Two new events are introduced at this level. The event *stuffing_ choices* consists of adding a corrupt choice by changing the variable *alone_ corrupt_ choices*, while the event *corrupt_ sig_ only* adds a corrupted signature by changing the variable *alone_ corrupt_ sig*. The variable *alone_ corrupt_ choices* is a subset of all choices, while the corrupted signatures are a subset of *Sig*.

Refinement 3 - introduces the main actor in the system i.e: the voter (elector). Voters having voted correctly become registered voters in the variable *honest_ voters*. Dishonest voters are registered in the variable *dishonest_ voters*. Voters who voted correctly can impersonate other voters in order to vote for them (or steal their vote). The honest voters are linked to their correct signatures via the variable *honest_ voters_ sig*, and an honest voter can have only one

signature at a time, and vice versa; thus, a correct signature is assigned to one and only one voter at a time ($inv3$). The corrupted signatures of the voters are defined in the relation between electors and the set of signatures (Sig) ($inv4$). Note that the domain and co-domain of these two variables have no common element ($inv5$ et $inv6$).

$$
\begin{array}{l}
inv5 : dom(corrupt\_sig\_voters) \cap dom(honest\_voters\_sig) = \varnothing \\
inv6 : ran(corrupt\_sig\_voters) \cap ran(honest\_voters\_sig) = \varnothing \\
inv7 : honest\_know \in honest\_voters \rightarrow \mathbb{P}(valid\_choices) \\
inv8 : \forall v, elec, sig \cdot \left(\begin{array}{l} v \in valid\_choices \wedge elec \in honest\_voters \wedge sig \in valid\_sig \\ \wedge elec \mapsto sig \in honest\_voters\_sig \wedge sig \mapsto v \notin rec\_votes \end{array} \right) \\
\qquad \Rightarrow elec \mapsto \{v\} \notin honest\_know) \\
inv12 : \forall elec1, elec2, v1, v2 \cdot \left(\begin{array}{l} elec1 \in dom(honest\_know) \wedge honest\_know(elec1) = v1 \\ \wedge elec2 \in dom(honest\_know) \\ \wedge honest\_know(elec2) = v2 \wedge elec1 \neq elec2 \end{array} \right) \\
\qquad \Rightarrow (\forall vx \cdot (vx \in v1 \Rightarrow vx \notin v2)))
\end{array}
$$

Removing correct choices already made implies knowledge of the choices made by the honest voter. To ensure the secrecy of the vote, we add a variable representing voter knowledge ($inv7$). This variable is a total function of the voters who voted towards all subsets of choices. Secrecy is expressed by the invariant $inv12$ which states that the knowledge of how each known voter has voted is restricted to the voter themselves. Deleting a choice correctly implies that only the voter knows his/her choice and how they have voted. In contrast, an intrusion deletion does not require any knowledge of choices or how a voter has voted.

Refinement 4 - introduces authentication, which requires that the system has some guaranteed means of identification of voters. In our model, this is ensured by the following two constants introduced in the Event-B context $C0\_Recording$ ie: $Credentials\_assign$ and $Passwords\_assign$ that are defined as:
$Credentials\_assign \in Electors \rightarrowtail\!\!\!\!\!\!\rightarrow Credentials$, and $Passwords\_assign \in Electors \rightarrowtail\!\!\!\!\!\!\rightarrow Passwords$. The model consists of an assignment of credentials and passwords to eligible voters. Thus, each voter has his own identification that gives permission for access to his account that is by definition, unique to each voter . The authentication in our system consists of verification by introducing two events for this purpose. The first event allows electors who wish to establish a connection to access to their voting account (login), while the second allows the disconnection of a voter having already established a connection. An authentication modifies the variable $electors\_session$ introduced for this purpose. We note that this authentication allows access to the account for voting purposes, recording voting, etc .... The identification is expressed as follows:

$$
\begin{array}{l}
inv6 : \forall s, v \cdot \left(s \mapsto v \in valid\_votes \right) \\
\Rightarrow \exists elec, mdp, cred \cdot \left(\begin{array}{l} (elec \mapsto s) \in dom(voters\_hosting) \wedge elec \mapsto mdp \in Passwords\_assign \\ \wedge elec \mapsto cred \in Credentials\_assign) \end{array} \right)
\end{array}
$$

Authorization to vote requires that the elector entitled to vote has not yet voted. This check is performed by refinements 5 and 6. This stage distinguishs also intruders that try to establish a connection with stolen credentials and passwords. We thus introduce all variables that correspond to intruders' knowledge, and events for misused identity credentials, passwords, signatures and possibly removing choices already made by honest voters. Introducing these details means that the invariants $inv8$ and $inv12$ introduced in the previous refinement

are not sufficient. We need to express the requirement that knowledge of honest valid voters is not known by intruders. The following property expresses for instance the fact that honest choices are not known by the dishonest intruders.

$$inv14 : \forall elec1, v1 \cdot (elec1 \in dom(honest\_know) \wedge honest\_know(elec1) = v1) \Rightarrow \forall elec2 \cdot elec2 \in dom(dishonest\_know\_choice) \Rightarrow \forall vx \cdot (vx \in v1 \Rightarrow elec2 \mapsto \{vx\} \notin dishonest\_know\_choice)$$

Refinement 5 - considers location. In traditional systems, voting is done in a physical location or polling station, whilst in e-voting we replace the concrete locations with an abstract/virtual concept. Thus, we introduce in this refinement a new variable *regis_ votes_ offices*, which assigns each vote cast correctly to one and only one polling station.

$$inv1 : regis\_votes\_offices \in PollStation \leftrightarrow (rec\_votes)$$
$$inv2 : \forall ve \cdot (ve \in rec\_votes \Rightarrow (\exists h \cdot (h \mapsto ve \in regis\_votes\_offices))$$
$$inv3 : \forall v1, b1, b2 \cdot (b1 \mapsto v1 \in regis\_votes\_offices \wedge b2 \mapsto v1 \in regis\_votes\_offices \Rightarrow b1 = b2)$$

The recording of the vote is thus restricted by location; in other words, a restriction of authorizations for voters to cast a vote in the offices to which they were assigned. A list is established beforehand to assign eligible offices to voters; this being defined by the constant *voters_ hosting* in the Event-B context *C*0*_Recording* (*voters_ hosting* $\in$ *electoral_ roll* $\rightarrow$ *PollStation*). Thus, the events to record votes are reinforced by the guards:

$$grd8 : \exists sig \cdot (sig \in Sig \wedge heberg \in PollStation \wedge ((votant\_x \mapsto sig) \mapsto heberg) \in voters\_hosting)$$
$$grd9 : heberg \mapsto (s \mapsto v) \notin regis\_votes\_offices \wedge (s \mapsto v) \notin ran(regis\_votes\_offices)$$

and the next action is added to update the variable *regis_ votes_ offices*: *act*7 : *regis_ votes_ offices* := *regis_ votes_ offices* $\cup$ $\{heberg \mapsto (s \mapsto v)\}$. Thus, eligibility for honest voters is expressed as follows:

$$inv4 : (\forall s, v, h \cdot (s \mapsto v \in valid\_votes \wedge h \mapsto (s \mapsto v) \in regis\_votes\_offices \Rightarrow \exists elec \cdot (elec \in votant \wedge elec \mapsto s \in honest\_voters\_sig \wedge (elec \mapsto s) \mapsto h \in voters\_hosting)))$$

which expresses that for all correctly recorded choices (*valid_ votes*) in polling stations (*regis_ votes_ offices*), there exists an eligible voter having a valid signature (*honest_ voters_ sig* introduced in the third refinement), with an identical signature, previously registered in the system, that casts this said choice.

Refinement 6 - models the depedency between the choice offerred to, and taken by, the voters and the specific type of election/referendum being run; and the anonymity of this choice. The recording of a vote is preceded by the choice that can be made by an eligible elector. The choice of bulletins must be anonymous, which can be guaranteed by the use of envelopes, as is the case in classic voting. We introduce in this refinement several new variables that facilitate the modelling of envelopes during the vote recording process. The variable *valid_ envelopes* corresponds to the envelopes chosen by voters. A voter who took an envelope is added to the variable *voters_ envelopes*. Each valid choice is assigned to a single valid envelope. The choice of voter is made concrete by the event *choose*. To make a choice, a voter must have authorization for this action. The actions enabled by this event are guarded by the existence of the person who wishes to initiate a process of voting in the list previously established *electoral_ roll* and that no signature is yet registered to his vote.

Refinement 7 - Different elections have different modes/types of voting. For example, a *majoritarian voting* where a presidential candidate must be elected is represented by paper ballot where every candidate is the option to vote and each paper corresponds to one candidate (vote or poll). This constraint is shown in Event-B by the following constant: $axmt1$: $bulletins\_representatives \in Representatives \rightarrowtail Bulletins$ where *Representatives* corresponds to the set of all representatives needed for a specific election including designations that may be chosen by a voter. For instance, this set can contain: $candidat_1$, $candidat_2$, ..., $candidat_n$, $None\_of\_the\_above$, in the case of a presidential election. It may also contain *favorable*, *unfavorable*, if the choice in a referendum is an adherence to any law.
In the case of a *preferential voting* or *cumulative voting*, voters should make their choice on paper ballot, where all candidates are listed on all these papers. This choice corresponds to a preference order mentioned next to each candidate on the same paper ballot. This constraint corresponds to a Cartesian product presented as follows in the Event-B method: $axmt2$: $bulletins\_representatives = Representatives \times Bulletins$. These constraints situate our development and thus contextualize the proofs. We have shown that constraints rely on the static part in the system, and we qualify this as a ***context of constraints***. Each type of voting is defined in a different Event-B context. These two Event-B contexts extend the first one introduced in the beginning of this section ($C0\_Recording$) and are noted by $C1\_Recording\_T1$ et $C1\_Recording\_T2$. At this stage of refinement, the machine introduced in the previous refinement is refined into two different machines. This decomposition allows each machine to see a different Event-B context. Thus, the machine $M8\_Recording\_T1$, (respectively the machine $M8\_Recording\_T2$) sees the Event-B context $C1\_Recording\_T1$ (respectively the context $C1\_Recording\_T2$). In the following, we report on the development of the first type of voting.

Each voter who has selected a paper ballot is added in the variable *bulletins_ voters* with their own bulletin. This action is observed in the event *choose*. The selected paper ballot and the voter are added to the variable *bulletins_ voters*. This choice represents a ballot stored in the variable *ballots*. The voter puts one and only one name or paper (or candidate) in the ballot box. Therefore, one and only one ballot is sleeved in an envelope. The recorded bulletins are a subset of the set of *Bulletins* . This variable will serve us in the Event-B context corresponding to tallying. The variable *ballots_ offices* allows us to record the ballots per polling station. The casting of votes in the ballot boxes is made concrete by modifying the variable *rec_ votes* associated to a specific polling station in the recording event. It is based on the representatives indicated on the collected papers through the variable *collected_ bulletins_ representatives* that the affectation of voices to these representatives will be made.

Once this phase is finished, i.e. : the counter to express time comes to the end, the tallying phase can begin using the results obtained. In the following section we explain the formal dependency mechanism used to model the transfer of results.

4 Dependency Relationship between Voting Phases

The data provided for the B contexts of this phase are deduced from the first phase corresponding to a validation of both B contexts by machines from the first phase. We detail in the following the tallying of the first type of voting that corresponds to the machine $M8\_Recording\_T1$.

This Event-B context includes all elements defined in the first phase, namely, $C1\_Recording\_T1$. The context $C0\_Tallying\_T1$ extends $C1\_Recording\_T1$ and contains, in addition to elements defined in $C1\_Recording\_T1$, some of the variables defined in the machine $M8\_Recording\_T1$ which are defined as constants in the present context listed in twenty two axioms. For instance, *collected_ bulletins_ representatives*,*rec_ votes*, *valid_ sig* ....

Abstract Model: Phase of Tallying - The desired termination property for this phase of any voting protocol is identical for all types of voting, thus, the first abstract model $M0\_Tallying$ is common to both types of voting that we introduce in this phase. This model describes the counting via three events *tally*, $finish$ and *maintain*. The only variable introduced at this level is a boolean which verifies whether counting is complete. The event *tally* is observing the value of this variable which is "false" in its guard, and does not perform any action in this machine.

Refinement 1: Phase of Tallying - In this machine which refines the $M0\_Tallying$ machine, the tally is done at the specific polling station. The variable *correct_ result_ office* characterizes the representative's scores per polling station ($inv1$: *correct_ result_ office* $\subseteq$ *PollStation* $\times$ (*Representatives* $\times \mathbb{N}$)). In each polling station, the scores of each representative are unique:

$$inv2 : \forall h, r, x1, x2 \cdot \left(\begin{array}{l} h \in PollStation \wedge r \in Representatives \\ \wedge\, h \mapsto (r \mapsto x1) \in correct\_result\_office \\ \wedge\, h \mapsto (r \mapsto x2) \in correct\_result\_office \end{array} \right) \Rightarrow x1 = x2$$

At initialization, no representative has received any votes. The counting of the voters' choices requires representatives who are registered in the envelopes (*destroyed_ envelopes_ representatives*), and the ballots recorded per polling station (*destroyed_ ballots_ office*). In addition, one must know the representatives of registered ballots (*destroyed_ bulletins_ representatives*). These variables can be seen as ***"copies"*** of constants defined in the Event-B context seen by the present machine. To verify that all registered bulletins were correctly counted, after the end of tallying, we must ensure that all voters who have signed have a bulletin that has been counted, and vice versa, all counted bulletins correspond to choices of voters who indeed signed. We introduce a new variable *counted_ bulletins_ representatives* that contains the bulletins, effectively counted. As all bulletins contain at most one representation (*bulletins_ representatives* $\in$ *Representatives* $\rightarrowtail$ *recorded_ bulletins*), this guarantees verifiability of the dynamic behaviour of the system.

$inv3 : destroyed\_envelopes\_representatives \subseteq envelopes\_representatives$
$inv4 : destroyed\_bulletins\_representatives \subseteq (collected\_bulletins\_representatives \rhd (ran(valid\_envelopes \lhd ballots)))$
$inv5 : destroyed\_ballots\_office \subseteq ballots\_offices$
$inv6 : counted\_bulletins\_representatives \subseteq (collected\_bulletins\_representatives \rhd (ran(valid\_envelopes \lhd ballots)))$
$inv7 : counted\_bulletins\_representatives \cap destroyed\_bulletins\_representatives = \varnothing$

This property is true only if the votes were recorded correctly without being corrupted. Verifiability is also expressed in the context seen by the present machine[3]. Other properties are also expressed to say, for instance, that if there exists a corrupt paper ballot, then these are not counted. At initialization, all variables are initialized with the values of the corresponding constants, with the exception of the variable *counted_ bulletins_ representatives* which is initialized to the empty set. The variable *checked* introduced in the abstract model of the same phase will be maintained in this machine, and its refinement. The tally counts all correctly recorded choices in polls (*destroyed_ bulletins_ representatives* that is a copy of the collected choices in *collected_ bulletins_ representatives*). The variable *checked* is a boolean initialized to false, that asserts that the tally can continue as long as there exist ballots not yet counted. This property is expressed by the variant of this machine, and guarantees convergence of the tallying process.

Refinement 2: Phase of Tallying - Finally, the tally for each representative corresponds to the number of total votes (sum of voices by office). This refinement introduces the total computation of voices of each representative saved in the variable *global_ result*: $inv1 : global\_result \in Representatives \rightarrow \mathbb{N}$. Each representative has zero votes/voices at the initialization, and the action incrementing the total voices of each representative is added to the same event for counting.

Condition for Dependencies - We recall that the dependencies between two parties (two models) $\mathcal{M}_1$ and $\mathcal{M}_2$ are defined by: *i)* the B contexts seen by the first machines are also seen by the machines defined for the second component; *ii)* a transformation of a some variables of the first model $\mathcal{M}_1$ into constants in the target model $\mathcal{M}_2$; *iii)* the predicate characterizing the termination property of the first model satisfies the constraints defined in the B context of the target model.

The stability in the first model is defined over traces generated from the machine in this model. This modelling reflects the fact that at the end of the first phase, no changes can be made on these elements as variables, because these variables in the phase of registration maintain their values at the termination. Therefore, we can define them as constants in this Event-B context. A vote is validated only when all the constraints defined in this Event-B context are valid. The validation of such constraints is based on facts or data generated during the recording phase. This implies the existence of states in the model $M8\_Recording\_T1$ satisfying these constraints that we call ***context deduced or combination of situations and constraints***. The satisfaction of axioms thus defined, particularly

[3] Note that in the real development this property is more complicated than the one presented in this document. The full, more complex model, can be obtained from the authors on request.

the axioms **dep_axm23** and **dep_axm24**, expresses the ***"initial configuration"*** of this phase of the vote: $C0\_Tallying\_T1(s2, c2) \wedge Init_2$, where s_2 and c_2 are respectively, sets and constants of the B context $C0\_Tallying\_T1$.
This relationship expresses a dependency between these two components. In particular, the two axioms **dep_axm23** and **dep_axm24** should be validated by properties over values of state variables of the previous phase.
To express the validation of these constraints, we introduce the constant *valide*. This constant also depends on the state the machine 8 of the registration phase. The states that validate these constraints are the states which, in addition to satisfying the axioms **axm1** ...**axm22**, must also fulfill the conditions defined in the axiom **dep_axm24** which expresses constraints, such as: (1) the closing time of polls has arrived: $timer = end\_time$; (2) no corrupt signature has been recorded: $alone\_corrupt\_sig = \varnothing$; (3) no corrupt choices assigned to an envelope have been recorded: $corrupt\_choices\_envelopes = \varnothing$; (4) choices and signatures are registered in the polls provided the voters who made these choices have signed at offices where they were registered to vote: $\forall\ s, v, h \cdot (s \mapsto v \in rec\_votes \wedge h \mapsto (s \mapsto v) \in regis\_votes\_offices \Rightarrow \exists\ elec \cdot ((elec \mapsto s) \mapsto h \in voters\_hosting))$; (5) the number of correct votes is the same as the number of recorded envelopes: $correct\_choices\_envelopes \in valid\_choices \rightarrowtail\!\!\!\!\!\!\twoheadrightarrow valid\_envelopes$; and (6) a recorded vote (with valid choices and signatures) can not belong to two different offices:
$(\forall\ v1, b1, b2 \cdot (b1 \mapsto v1 \in regis\_votes\_offices \wedge b2 \mapsto v1 \in regis\_votes\_offices \Rightarrow b1 = b2))$.

The designed patterns have generated 1317 proof obligations, among which 757 are discharged in an non-automatic manner. Non-automatic proof obligations are related to properties using universal quantification. The instantiation of the patterns consists in specifying values of sets in B contexts, which does not give rise to additional proof obligations and in introducing other refinements for specific needs of designers.

5 Conclusion and Future Work

5.1 Contributions: contexts, refinements and dependency

Our overall contribution is to illustrate a formal method for combining contexts, refinements and dependency composition in a coherent and reusable manner. Two main voting families appear in our development. However, the specific family remains implicit. It follows that the interpretation of the results is not taken into account in our modelling. The certification of models needs to describe the voting method in order to make a decision. Such an interpretation thus depends on the context in which the proofs are made. Contexts are formal objects [17] based on McCarthy's principle that contexts are constructed incrementally from previous ones, which corresponds to *"context lifting"*. The situation appears as a new parameter in the predicates and thus, predicates depend on a situation. A lifting involves situations or times. In the Event-B formalism, situations are states and constraints are static properties defined in Event-B contexts. Thus,

the dependency relationship in this formalism is defined as a combination of states and constraints. The dependency is a measurable relationship taking values from situations facts and giving rise to new proof obligations. Such a principle represents a duality to the principle of invariance in Event-B machines, claiming that states are constrained by invariants in order to establish safety in a proof system.

5.2 Future work: Security Issues

Security is an important issue for ensuring reliable operation and protecting the integrity of stored information to guarantee a trustworthy e-voting system [6]. These are based on a systematic engineering approach achieved by the identification, detection and correcting security risks and threats, requirements and recovery strategies [13]. Thus, validation of the assumptions made by designers is performed on threat modelling attached to their contextual information to safeguard the system against unauthorized modification of data, or disclosure of information. Deeper analysis of the security in an e-voting system relies also on identifying *assets* to determine answers to questions about what the system is designed to protect, and from whom [19]. Our modelling deals only with voters. To target a particular system, it would therefore be necessary to integrate the different assets. This can be achieved by defining a set *Assets* in the Event-B context of the recording phase, and all these actors will be constants included in this set.

Our development can be combined with the already realized models of N. Benaïssa [4,3] that deal with the key establishment properties for the preparation phase. His works deal with the authentication properties, as well as the key establishment goals combined with the attacker's knowledge. The authentication models can be reused as input provided from the preparation phase of the vote to the voting phase in our development as a result via the dependency mechanism. We can also consider the probabilistic approaches such as blind signatures, mix nets or encryption schemes.

References

1. Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.
2. K. Jon Barwise. Conditionals and conditional information. In Elizabeth Traugott, Alice ter Meulen, Judy Reilly, and Charles Ferguson, editors, *On Conditionals*, pages 21–54. Cambridge University Press, Cambridge, England, 1986.
3. Nazim Benaissa. Modelling Attacker's Knowledge for Cascade Cryptographic Protocols. In Egon Börger, Michael Butler, Jonathan P. Bowen, and Paul Boca, editors, *First International Conference on Abstract State Machines, B and Z - ABZ 2008*, volume 5238, pages 251–264, London, United Kingdom, September 2008. Springer.
4. Nazim Benaïssa and Dominique Méry. Proof-based design of security protocols. In *Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings*, pages 25–36, 2010.

5. Dines Bjorner. *Software Engineering 3 Domains, Requirements, and Software Design*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. ISBN: 978-3-540-21151-8.
6. Lichun Chiang. Trust and security in the e-voting system. *Electronic Government, an International Journal*, 6(4):343–360, 2009.
7. Véronique Cortier, David Galindo, Stephane Glondu, Malika Izabachene, et al. A generic construction for voting correctness at minimum cost-application to helios. *IACR Cryptology ePrint Archive*, 2013:177, 2013.
8. Richard Dapoigny and Patrick Barlatier. Modeling contexts with dependent types. *Fundam. Inform.*, 104(4):293–327, 2010.
9. 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.
10. 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.
11. Thai Son Hoang, Andreas Furst, and Jean-Raymond Abrial. Event-b patterns and their tool support. *Software Engineering and Formal Methods, International Conference on*, 0:210–219, 2009.
12. Souad Kherroubi and Dominique Méry. Contextualisation et dépendance en Event-B. In Akram IDANI and Nikolai KOSMATOV, editors, *Approches Formelles dans l'Assistance au D'éveloppement de Logiciels, AFADL 2017*, 2017.
13. Gerald Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998.
14. Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 186–200, 2005.
15. Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.
16. Gary T. Leavens, Jean-Raymond Abrial, Don S. Batory, Michael J. Butler, Alessandro Coglio, Kathi Fisler, Eric C. R. Hehner, Cliff B. Jones, Dale Miller, Simon L. Peyton Jones, Murali Sitaraman, Douglas R. Smith, and Aaron Stump. Roadmap for enhanced languages and methods to aid verification. In *GPCE*, pages 221–236, 2006.
17. John McCarthy. Notes on formalizing context. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, pages 555–560, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.
18. Dominique Méry and Mike Poppleton. Towards An Integrated Formal Method for Verification of Liveness Properties in Distributed Systems. *Software and Systems Modeling (SoSyM)*, December 2015.
19. Suvda Myagmar, Adam J Lee, and William Yurcik. Threat modeling as a basis for security requirements. In *Symposium on Requirements Engineering for Information Security (SREIS)*. IEEE, August 2005.