

Specification and Analysis of the Electronic Voting Process for the ES&S Voting System

Komminist Weldemariam
DISI, University of Trento, Italy
sisai@fbk.eu

Richard A. Kemmerer
CS Dept, UC Santa Barbara, USA
kemm@cs.ucsb.edu

Adolfo Villafiorita
Fondazione Bruno Kessler (FBK-irst), Italy
adolfo@fbk.eu

Abstract

Electronic voting systems are a perfect example of security-critical computing. One of the critical and complex parts of such systems is the voting process, which is responsible for correctly and securely storing intentions and actions of the voters. Unfortunately, a recent study revealed that various e-voting systems show serious specification, design, and implementation flaws.

Formal specification and verification can greatly help to better understand the system requirements of e-voting systems by thoroughly specifying and analyzing the underlying assumptions against security specific properties.

This paper presents the specification and verification of the electronic voting process for the Election Systems & Software (ES&S) system. We used the ASTRAL language to specify the voting process of ES&S machines and the critical security requirements for the system. Proof obligations that verify that the specified system meets the critical requirements were automatically generated by the ASTRAL Software Development Environment (SDE). The PVS interactive theorem prover was then used to apply the appropriate proof strategies and discharge the proof obligations.

1 Introduction

Electronic voting systems are increasingly replacing the traditional paper-based voting systems. These systems can make the voting process more convenient and may therefore lead to improved turnout. Electronic recording and counting of votes could be faster, more accurate and less labor intensive.

However, as witnessed in [14, 7, 4, 20, 22, 26], several e-voting systems share critical failures in their design and implementation, that render their technical and procedural controls insufficient to guarantee trustworthy voting.

There are a number of ways in which the integrity and assurance of a complex system's correct behavior with

respect to a specification can be achieved. Among them, formal verification has been shown to improve the security and quality of complex systems. See, for instance, [10, 35, 3].

The goal of this paper is to demonstrate how the use of formal techniques can help to ensure fair elections. We argue that we can build better e-voting systems by getting a more profound knowledge of the security risks and possible countermeasures both at the system and at the procedural level. We do so by presenting the specification of the voting process for the Election System & Software (ES&S) voting system as documented in the EVEREST project report [21]. We treated the ES&S voting system as a complex, real-time embedded system, consisting of a direct recording election machine (DRE), a real-time audit log printer (RTAL), a personalized election ballot (PEB), and a Compact Flash Card (CF). We mapped each of these components to ASTRAL [8] process instances, and we also specified security critical requirements to prove the correctness and integrity of the election results. The consistency of the specification was validated using the ASTRAL validation engine, and PVS [23] proof obligations were automatically generated by the ASTRAL Software Development Environment (SDE). These proof obligations verify that the specified system meets the critical security requirements. The PVS interactive theorem prover was then used to apply the appropriate proof strategies and discharge each of the proof obligations.

This paper is organized as follows. The next section presents the background material for the ES&S system. A short description of ASTRAL is given in Section 3, and the ASTRAL specification of the system and its critical security requirements and the status of the PVS formal verification is presented in Section 4. In Section 5 we discuss related work. Finally, we draw conclusions and discuss future work in Section 6.

2 Electronic Voting Systems

2.1 ES&S Voting System Components

For the purposes of this work (see [21] for a more detailed and complete view), the ES&S voting system is composed of:

- *DRE*: Direct Recording Electronic voting machine, called the iVotronic. It is equipped with a touch-screen where the voter casts his/her votes. The information shown by the touch-screen changes in real-time to match the voter's choices. The iVotronic also stores the audit data.
- *RTAL*: Real-Time Audit Log Printer, which performs the function of a VVPAT (Voter-Verified Paper Audit Trail) for the ES&S system. It produces a paper-based record of the choices selected by the voter. The RTAL is plugged into the DRE and the paper record is viewable by the voter. The voter's choices are under a transparent cover so that they cannot be modified other than through the normal voting process.
- *PEB*: Personalized Electronic Ballot. This is a device used by the poll worker to load a ballot, initialize the next ballot, and collect tabulated data and audit information. Each time a PEB is inserted, its authenticity is checked by the DRE using a four-digit code (election qualification code, EQC), which is assigned prior to election day.
- *CFC*: Compact Flash Card. This device holds files too large to fit in the PEB and also audit data. The card must be present to open and close the polls. At poll closing, the audit data is automatically dumped into the card.

2.2 Voting Process for a DRE based System

In this section we present a high-level overview of running an election using the ES&S system. We skip the description of the election preparation phase (e.g., the pre-election day operations checklist, obtaining an EQC code, qualifying PEBs, and clearing and testing voter terminals), which are described in [21]. Instead, we assume that these operations have been carefully and correctly done at the central location.

Prior to opening the polls, a poll worker unpacks and sets up the DRE and plugs in the RTAL printer and power cables. Poll workers must also ensure that a properly programmed CF card is installed before powering on the DRE.

A Master PEB is inserted into the terminal to load the ballot and later to open the terminal for voting. The same

master PEB must be used to close the terminal after the polls have closed. Removing the PEB turns the terminal's current mode to sleep mode.

Once the polls are opened, a poll worker initializes the ballot for a qualified voter by inserting a supervisor PEB, which can be the same Master PEB used to open the polls, into the machine. The terminal mode changes from sleep to poll worker mode, the EQC code of the PEB is checked, and the ballot is initialized, provided that the EQC of the PEB matches with the one the terminal is configured for. The poll worker removes the supervisor PEB and leaves the terminal for the voter.

After the ballot is activated, the machine takes the voter through each contest. DRE machines automatically forbid overvoting, but not undervoting. When a voter selects or cancels a candidate for a particular contest, an appropriate indication is printed on the RTAL record. If the voter selects a candidate, the RTAL record is marked as "Selected" and scrolled out of sight; otherwise, it is marked as "Canceled" and scrolled out of sight. The voter is eventually given the opportunity to review his ballot, and if the voter commits to it (confirms it), it is recorded to local storage. The process continues in this way for all qualified voters.

After the official poll closing time is reached and there is no qualified voter waiting in line, the poll worker inserts the master PEB to collect and store tabulated data, copies of the ballot image, and some other information. Upon closing the terminal, the DRE firmware automatically uploads the audit data onto the CF card. The results tape from the RTAL is also collected. The results tape, CF card, and master PEB from each polling place are then returned to election central.

2.3 Informal Requirements for the ES&S System

Fairness and security of electronic elections depend upon a careful allocation of requirements to the procedures and to the systems used. In fact, on the one hand, the correct behavior of the electronic systems can be guaranteed only when they are used according to their operating specifications. This has to be guaranteed by the procedures and the people responsible for executing them. For example, there is no way for an e-voting machine to prohibit the same person from casting multiple ballots, if the poll workers enable the machine for voting multiple times. This behavior can be prevented (or revealed after the election) only by enforcing and verifying the procedures that the poll workers are supposed to follow.

In contrast, there are certain fundamental properties that the procedures can assure only up to a point. In this case, the e-voting systems must guarantee that these properties are satisfied. Using the example we just made,

the machine must ensure that a voter can cast at most one vote when the poll workers follow the prescribed procedures.

Analyzing such requirements and such allocation of requirements is therefore important in two respects. First, it helps to ensure that the systems meet the necessary reliability and dependability goals. Second it helps to better understand how a different allocation of requirements between systems and procedures could improve the overall security of the election process.

We did this work for the ES&S system. A number of requirements that the ES&S system must satisfy are enumerated in the ES&S system manual [13] and a corresponding video¹, which describes how the system works on election day. In what follows we present the most important critical requirements that the system must satisfy.

Requirements for the DRE

A correctly functioning DRE must satisfy the following properties.

- *D-Req1*: The same CF card must be present throughout the voting session;
- *D-Req2*: The RTAL must be connected to the DRE throughout the voting session;
- *D-Req3*: The DRE must authenticate the PEB using the EQC, and the same master PEB must be used to open and close the terminal;
- *D-Req4*: Once the ballots are loaded into the DRE prior to starting the voting process, they should not change during the entire course of the voting process;
- *D-Req5*: Display screens presented to the voter must accurately reflect the ballot downloaded from the PEB and the selections made by the voters;
- *D-Req6*: The DRE terminal only allows two valid actions for the voter until he/she reaches the final review (vote summary) screen: (1) select or cancel a candidate on the screen, or (2) move forward or backward through the ballot;
- *D-Req7*: For each valid voter action (i.e., starting to vote, making a select or cancel, and finishing a vote) the DRE must enable the RTAL to record the action on the RTAL tape accordingly;
- *D-Req8*: The DRE must automatically forbid an overvote;

- *D-Req9*: The DRE must report undervoted races, if they exist, and the review screen must display the message “BALLOT NOT COMPLETED”;
- *D-Req10*: When the voter confirms his/her ballot, the ballot images recorded in the local storage must correctly reflect the selections made by the voter;
- *D-Req11*: The DRE terminal should start chirping if there is no input from the voter for 10 minutes since the last input but not after s/he confirmed.

Requirements for the RTAL

The RTAL must satisfy the following properties:

- *R-Req1*: The RTAL should scroll up a minimum distance after the summary has printed;
- *R-Req2*: The RTAL must update the paper tape after the voter pushes the start button, makes a choice (select or cancel), confirms a vote, or when the poll worker rejects the ballot of a fleeing voter.

Requirements for the PEB

The PEB must satisfy the following properties:

- *P-Req1* The election-specific secret code (EQC), which is a 32-bit (4 digit) code, must be present on a PEB and must always match with the one stored inside the DRE; otherwise, the PEB should be rejected by the DRE terminal whenever the poll worker attempts to insert it;
- *P-Req2* At the end of the election, the copy of the ballot images downloaded from the DRE must be the same as the ballot images that were loaded into the DRE prior to starting the election.

Requirements for the CF Card

The CF card should satisfy the following property:

- *C-Req* The poll closing procedures must copy the audit information (such as the event log) accumulated in the local storage to the CF card.

Requirements for the System as a whole

The following global properties must be ensured by the system components all together:

- *G-Req*: No discrepancy should be observed among the following: (1) the individual cast ballot records (or ballot images) recorded by the machines; (2) the summary tape generated on Election Day at the close of polls on individual machines; (3) the totals

¹http://www.essvote.com/HTML/voter_outreach/ivotronic_flash.html

that were accumulated and reported by the DRE and RTAL.

The above requirement can further be classified into the following integrity of election result requirements:

- *G-Req1*: The vote entries printed on the RTAL tape during the election must be equal to the cast ballot records plus the rejected vote in the DRE;
- *G-Req2*: After the voting is closed, the results downloaded into the master PEB must be equal to the sum of the results collected from each DRE; furthermore, it must be equal to the sum of the printed paper tapes from all RTALs;
- *G-Req3*: The number of fleeing voters recorded in the audit log file, which is downloaded into the CF card, must be equal to the number of rejected ballots printed on the RTAL tape;
- *G-Req4*: The undervoted races in the audit log file, which is downloaded into the CF card, must be equal to the undervoted races that have been reported on the RTAL tape.

In Section 4.2, the above requirements are converted into ASTRAL invariants, schedules, and constraints.

3 The ASTRAL language

ASTRAL [8] is a high-level formal specification language designed for reactive systems. The language constructs allow one to build modularized specifications of complex systems using state machines. ASTRAL provides a mechanism for specifying critical system requirements as first order formulas, and a formal proof system for proving that the system actually meets the stated requirements.

An ASTRAL specification of a system consists of a global specification and process specifications. The global specification contains declarations of the process instances, global constants and non-primitive types (which may be shared by process instances), and system level critical requirements.

An ASTRAL process specification presents an abstract model of what constitutes the process (types, constants, variables), what the process does (state transitions), and the critical requirements the process must meet. The process being specified is thought of as being in various states, with one state differentiated from another by the values of the state variables, which can be changed only by means of state transitions. A transition is modeled by entry and exit conditions, and a non-zero duration is assigned to each entry/exit pair. Specification exceptions are handled explicitly by adding except/exit

pairs in addition to the normal entry/exit pairs. Transitions are executed as soon as the entry conditions are satisfied assuming no other transition for that process instance is executing.

Every ASTRAL process can export both state variables and transitions; as a consequence the former are readable by other processes while the latter are executable from the external environment. Interprocess communication is accomplished by broadcasting the value of exported variables, as well as the start and end times of exported transitions. In addition to specifying system state (through process variables and constants) and system evolution (through transitions), an ASTRAL specification also defines system critical requirements and assumptions on the behavior of the environment that interacts with the system. The behavior of the environment is expressed by means of environment clauses, which describe assumptions about the pattern of invocation of external transitions. Critical requirements are expressed by means of invariants and schedules. Invariants represent requirements that must hold in every state that may be reached from the initial state, no matter what the behavior of the external environment is, while schedules represent additional properties that must be satisfied provided that the external environment behaves as assumed.

4 Specification of the ES&S Voting System

The complete ASTRAL specification of the ES&S is approximately 30 pages long. In this section, therefore, we are able to present only a sampling of the specification that, we believe, provides the flavor of the work.

In the ES&S system specification presented in this section: the DRE must authenticate each PEB using its four digit EQC, it should correctly handle vote selection (specifically: it must process and store vote selection and alert the RTAL to scroll the paper and print the information), and it should periodically check for the existence of the CF card. The RTAL printer should be continuously working during the election period. These all collaborate to maintain the integrity of the election results. In fact, the integrity of the election results depends heavily on the integrity of the software and firmware that runs the central election management system (EMS) and the hardware used (as discussed in [21]). However, this is largely dependent upon a particular implementation and is not in the scope of this specification. Moreover, audit logs serve a vital purpose, as they can alert an auditor of suspicious or uncommon events that occurred, which could indicate the presence of malicious intent against the system. Because of this, it is critically important that an auditor is completely confident that the information retrieved from the audit logs is complete and accurate. Therefore, the security properties specified in Section 4.2

mainly concentrate on the integrity of election results.

4.1 ASTRAL Specification of the ES&S

We formulate each component of the electronic voting process as an ASTRAL process instance (See Figure 1).

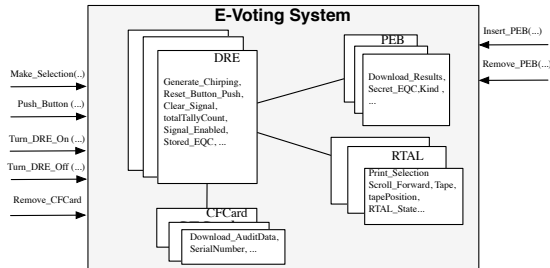


Figure 1: A simplified view of ES&S voting system

Four process types are declared in the global specification of the ASTRAL model of the ES&S:

```

PROCESSES
the_DRE: array [1..Number_Of_DRE]
  of DRE_Process,
the_RTAL: array [1..Number_Of_RTAL]
  of RTAL_Process,
the_PEB: array [1..Number_Of_PEB]
  of PEB_Process,
the_CFCard: array [1..Number_Of_CFCard]
  of CFCard_Process

```

The above declaration indicates that there are `number_of_DRE` DRE instances (we assume one per polling station) of type `DRE_Process` in the model, where `number_of_DRE` is a global constant.

Besides primitive type declarations, ASTRAL allows one to make custom type declarations (i.e., defined types), like in the following snippet of specification:

```

TYPE
PEB_ID: TYPEDEF p: ID ( IDTYPE ( p ) =
  PEB_Process ) ,
PrintValue/* Unspecified type that
  represents the print format
  of the RTAL.*/,
StoreBallot,
MessageType,
Race,
Races IS SET OF Race,
Candidate,
Candidates IS SET OF Candidate,
Race_Candidate_Pair IS STRUCTURE OF
  (Contest: Race, Nominees: Candidates ) ,
Ballot IS SET OF Race_Candidate_Pair,
Decision IS ( Selected, Canceled ) ,

```

```

Button IS ( START_BUTTON, BACK, CANCEL,
  CAST, CLOSE, CONFIRM, EXIT, NEXT,
  RESET, REVIEW ) ,
[...]
```

The `DRE_ID` line declares `DRE_IDs` to be exactly those ids that are for process instances of type `DRE_Process`. (Similarly defined for the rest of the processes.) The `Decision` and `Button` are enumerations, that represent, respectively, the voter's decision on a contest and the buttons that can be used to interact with the touchscreen.

The `Ballot` type is a representation of the “ballot images,” as a collection of race-candidate pairs.

The first two constants declared in the global specification associate each DRE with a unique Compact Flash Card and RTAL printer. The third one represents the print format on the RTAL tape when the voter selects or cancels a particular candidate. In addition to printing vote selections, the RTAL also prints start and summary information for each voter session, which is represented by the last constant.

```

CONSTANT
  Installed_CFCard( DRE_ID ): CFCard_ID,
  Plugged_In_RTAL( DRE_ID ): RTAL_ID,
  Make_Print_VoteEntry(Name, Title,
    Decision): PrintValue,
  Make_Print_Info ( MessageType )
    : PrintValue

```

4.1.1 Modeling the DRE Process

The ES&S DRE device is modeled in ASTRAL by the process type `DRE_Process`. The initial clause of the model states that a Compact Flash Card is inserted in the machine and that a unique RTAL printer is attached to the DRE:

```

INITIAL
EXISTS f: CFCard_ID
  ( f = Installed_CFCard ( Self )
    & Which_CFCard_Installed = f
    & CFCardSerialNumber = f.SerialNumber )
& EXISTS r: RTAL_ID
  ( r = Plugged_In_RTAL ( Self )
    & Which_RTAL_Plugged_In = r )
& CFCard_Installed = TRUE
& RTAL_Plugged_In = TRUE

```

The keyword `Self` is used to refer to the process id of the process instance in which it appears. Every process instance has a unique process id.

The DRE machine stores vote records locally and automatically forbids overvotes, but not undervotes. The number of candidates currently selected for a particular

race and the total number of votes for a particular candidate in a race are modeled with the following two variables:

```
VARIABLE
  Number_Of_Selected ( Race ),
  TotalTallyCount (Candidate, Race)
                    : Non_Negative
```

Communication with the RTAL process is modeled through the following variables:

```
VARIABLE
  Signal_Enabled: Boolean,
  Which_Signal : SignalType
```

where the first variable signals that the DRE is sending data to the printer and Which_Signal carries the kind of information to be printed (e.g., is the print information a start vote session message or a vote selection).

To model permissible operations on the DRE machine, it is also necessary to capture the phases of the election and the various modes of the terminal during election day. We use the following variables:

```
VARIABLE
  Which_Phase: Election_Phase,
  Terminal_Mode: Mode,
  DRE_State: Terminal_State,
```

that indicate, respectively, the phase of the election (pre-voting, during-voting, and post-voting), the terminal mode (poll worker, voter, sleep, or chirping), and the state of the poll (opening, opened, closing, or closed). The last two variables are only meaningful during the actual voting process (i.e., Which_Phase = During_Voting).

When a voter casts a vote, s/he is actually interacting with the system by navigating from one screen to another using an appropriate button. We model such interaction by assigning an integer number to each screen shown to the voter and by defining specification functions that take as input a screen number and return the information to be displayed and the buttons available.

```
VARIABLE
  scrNumber : Screen_Number,
  scrName : Screen_Name,
  Screen_Buttons ( Integer ): Buttons,
  Display ( Screen_Number ) : Screen
```

In the above declaration the Display variable holds the state of the screen as it is to be shown to the voter while s/he is voting, e.g., if the voter is in one of the race screens then the value of the Display contains the candidates of that race with appropriate button(s) displayed on it.

The behavior of the DRE is modeled by ASTRAL transitions. For instance the following specification models the activation of the machine for the election:

```
TRANSITION Turn_DRE_On
ENTRY [TIME : T_D_Dur]
  Terminal_Mode = Deactivate
  & DRE_State = Initial_State
  & Which_Phase = Pre_Voting
  & EXISTS card: CFCard_ID (
    card.SerialNumber = CFCardSerialNumber
    & Which_CFCard_Installed = card )
  & CFCard_Installed
  & ~DRETurnedOn
EXIT
  DRETurnedOn
```

The entry assertion for this transition specifies that the terminal is currently deactivated and in its initial state, that the election is not yet started (i.e., the current phase of election is “pre-voting”), that a properly programmed Compact Flash Card is already installed (this is done at Election Central and a tamper-evident security seal is usually placed over the CF slot), and that the DRE is currently off. The exit assertion for the transition indicates that DRE machine is now turned on for the next operation.

An ES&S DRE requires the insertion of a qualified PEB device in order to allow various operations to run the election. These operations include loading the appropriate ballot, opening or closing the polls, initializing the ballot, collecting results, and performing various administrative tasks. All these operations require that the PEB has been inserted in the iVotronic terminal. The insertion of a PEB is specified as follows (only the entry assertions are shown):

```
TRANSITION Insert_PEB ( p: PEB_ID )
ENTRY [ TIME : I_P_Dur1 ]
  /*Case1: To load the ballots,
  prior to start election.*/
  MachineTurnedOn
  & Stored_EQC = p.Secret_EQC
  & p.Kind = Master
  & ~PEB_Inserted
  & Terminal_Mode = deactivated
  & Which_Phase = Pre_Voting
  & DRE_State = Initial_State
  & ~Ballot_Loaded
  & FORALL R: Race
    ( Race_Candidates ( R ) = EMPTY )
EXCEPT [ TIME : I_P_Dur2 ]
  /*Case2. To Initialize a ballot or to
  reset the Chirping terminal and
  reject the vote.*/
  MachineTurnedOn
  & Stored_EQC = p.Secret_EQC
  & ~PEB_Inserted
  & Which_Phase = During_Voting
  & DRE_State = Opened
  & ( Terminal_Mode = sleep_mode
    | Terminal_Mode = chirping )
```

```

EXCEPT          [ TIME : I_P_Dur3 ]
/* Case3: To close the terminal.*/
MachineTurnedOn
& Stored_EQC = p.Secret_EQC
& p.Kind = Master
& ~PEB_Inserted
& ~enabled_PEB_Download
& ~enabled_CFCARD_Download
& Which_Phase = Post_Voting
& ( Terminal_Mode = sleep_mode
  | Terminal_Mode = administrator )
& ~P.ResultDownload_Completed ( Self )

```

In each entry case, the first and second conjuncts state that the DRE should be on and that there is no PEB currently inserted, while the third conjunct (`Stored_EQC = p.Secret_EQC`) specifies the fact that election-specific secret EQC code of the PEB must match with the EQC code stored inside the DRE. The remaining conjuncts of each entry assertion capture three requirements from the ES&S documentation related to the different uses of the PEB during the election.

After the operations that required the insertion of the PEB are completed, the poll worker must safely remove the PEB. This is modeled by the following transition, which has three entry assertions each corresponds to the different usage of the PEB during the election process:

```

TRANSITION Remove_PEB ( p : PEB_ID )
ENTRY          [ TIME : R_P_Dur1 ]
/*Case1: Removing after the
ballot has been loaded.*/
Terminal_Mode = administrator
& Which_PEB_Inserted = p
& PEB_Inserted
& p.Kind = Master
& ( End ( Insert_PEB ) > 0
& Start ( Remove_PEB ) = 0 )
& Which_Phase = Pre_Voting
& Ballot_Loaded
& DRE_State = Opening
& Now - notify_time1 >=
  Change ( DRE_State )
EXCEPT          [ TIME : R_P_Dur2 ]
/*Case2: Removing the PEB after
loading or after correcting the
chirping terminal or PEB change
requested.*/
Terminal_Mode = administrator
& Which_PEB_Inserted = p
& PEB_Inserted
& ( End ( Insert_PEB ) >
  End ( Remove_PEB ) )
& Which_Phase = During_Voting
& ( ( Ballot_Initialized
  & End ( Insert_PEB ) >
  End ( Initialize_Ballot ) )
  | Vote_Canceled
  | ChangedToMasterPEB )

```

```

EXCEPT          [ TIME : R_P_Dur3 ]
/*Case3: Removing after downloading
the election result, copy of
ballot images, and audit log.*/
Terminal_Mode = administrator
& Which_PEB_Inserted = P
& PEB_Inserted
& p.Kind = Master
& Which_Phase = Post_Voting
& P.ResultDownload_Completed ( Self )
& Which_CFCARD_Installed.ADDownload_Completed
& DRE_State = Closing
& Now - notify_time2 >=
  Change ( DRE_State )

```

where `Start (Remove_PEB)` indicates the last time that the transition `Remove_PEB` was fired, and `End (Remove_PEB)` indicates the last time the transition terminated. The first part of the entry assertion specifies that the PEB is still inserted under the administrator mode. The three possible cases when the PEB can be removed are:

- the election has not been opened yet, the ballot for the current election is loaded in the iVotronic's local storage, the poll is on the way to be open (i.e., opening), and the PEB has remained inserted in the machine for at least `notify_time1`.
- the poll is closing, the PEB has remained inserted in the machine for at least `notify_time2`, the inserted PEB has collected both the election results and copies of the images of the ballots cast, and the audit data has been uploaded onto the installed CF card.

The third case, which is the insertion of the PEB during voting is omitted here.

The result of the first case (i.e., the normal case), is changing the state of the system to allow voting and putting the machine in sleeping mode, as encoded in the model by the following variables

```

~PEB_Inserted &
Which_Phase = During_Voting &
DRE_State = Opened &
Terminal_Mode = Sleep_Mode

```

The initialization of a ballot when a qualified voter comes is specified in the model by the transition

```

TRANSITION Initialize_Ballot
ENTRY          [ TIME : I_B_Dur ]
DRE_State = Opened
& Terminal_Mode = administrator
& EXISTS p: PEB_ID
  ( Which_PEB_Inserted = p )
& PEB_Inserted
& Proceed_Ballot_Init

```

```

& ~Ballot_Initialized
EXIT
FORALL R: Race
  ( Displayed_Candidates ( R ) =
    { SETDEF C: Candidate (
      C ISIN Race_Candidates (R) ))
& FORALL R: Race, C: Candidate
  ( C ISIN Displayed_Candidates ( R )
& ~Picked ( Candidate_Name ( C ) ,
  Race_Title ( R ) ) )
& FORALL R: Race
  ( Number_Of_Selected (R) = 0 )
& underVotedRaces = EMPTY
& Ballot_Initialized
& ~Proceed_Ballot_Init

```

The entry conditions specify that the poll is opened, the terminal is in administrative mode, the PEB is inserted, and the ballot has not been initialized for the voter who is ready to cast his vote. It should be noted that the voting procedure usually allows voting after scheduled poll closing time as long as a qualified voter is still in line. The exit condition specifies that the candidates to be displayed on the corresponding race screen are assigned, all the variable values from the last voter are reset —i.e., the Picked value for each candidate name of the current race, the number of selections for each race, and the temporary vote list are all reset. Therefore, the ballot is ready for the next voter, and the local variables Ballot_Initialized and Proceed_Ballot_Init are set to true and false, respectively.

In a touch-screen based voting system, a voter makes a choice or changes a previous choice by touching the candidate name on the display. In either case, the DRE must capture and process the touch correctly.

Make_Selection is an exported transition, which must be called by the voter.

```

TRANSITION Make_Selection ( cName: Name )
ENTRY [ TIME : M_S_Dur ]
  Which_Phase = During_Voting
  & Terminal_Mode = voter_mode
  & Race_Screen ( scrNumber )
  & currentRace = Which_Race(scrNumber)
  & EXISTS C: Candidate
  ( C ISIN
    Displayed_Candidates (currentRace)
    & Candidate_Name ( C ) = cName )
  & Display(scrNumber)=Display_Contest(
    Race_Title ( currentRace ),
    Displayed_Candidates(currentRace),
    Screen_Buttons ( scrNumber ) )
  & ~Signal_Enabled
  & Which_signal = NoSignal

```

specifies the occurrence of a screen touch on a particular candidate's name. On entry, the DRE checks that the voter is voting during voting period, the terminal is in voter mode, the current screen is a race screen displaying both the current race with its candidates and the button(s) required to navigate through the screen, the touched candidate cName belongs to the displayed candidates, and that the DRE is not currently sending a signal to the RTAL.

The variable Picked is used to determine whether the candidate has been previously selected. This variable will eventually be used to update the totalTallyCount for the selected candidate name cName when the ballot is confirmed. The exit assertion for the Make_Selection transition is

```

IF
  ~Picked' (cName, Race_Title(currentRace'))
THEN
  IF
    Number_Of_Selected' ( curScreenNum' ) <=
      Number_Of_Choices_Per_Race (
        curScreenNum' )
  THEN
    Number_Of_Selected ( currentRace' )
      BECOMES
    Number_Of_Selected' (currentRace' ) + 1
    & Picked (cName,Race_Title( currentRace'))
      BECOMES TRUE
    & Display ( scrNumber' ) BECOMES
    Update(Display' (scrNumber'),cName,Marked)
    & tempVoteRecord ( currentRace' ) BECOMES
    tempVoteRecord' ( currentRace' )
    UNION
    { SETDEF C: Candidate (
      Candidate_Name ( C ) = cName ) }
    & pickedName = cName
    & pickedValue = Selected
    & Signal_Enabled
    & Which_Signal = Vote_Signal
  ELSE
    Min_Display ( scrNumber' )
      BECOMES Display_Info(
        OVERVOTE_ATTEMPTED, NoButton)
  FI
ELSE /*for canceling case*/
  [...]

```

For the case where the voter is making a selection —i.e., Picked is false when the transition is fired— the following scenario occurs: as long as there is no overvote attempted the number of selections for this candidate for the current race is incremented by one, Picked is set to true, the current screen is updated, and cName is included in tempVoteRecord, which will be used to display the voter's final selection when the voter queries a preview. In addition, the

exported variables `pickedName`, `currentRace`, `pickedValue` and `Which_Signal` receive new values, and the signaling variable is set to true. This indicates that the RTAL can now print the selection expressed in these exported variables. Otherwise, the voter attempted to overvote and the DRE will display the appropriate message on the screen.

For the case where the voter is canceling a previous selection —i.e., `Picked` was true when the transition fired— the exit assertion specifies that the number of selected candidates for the current race is decremented by one, `Picked` is reset to false, and `cName` is removed from the `tempVoteRecord`. The rest of the variables are updated accordingly.

```
[...]
Number_Of_Selected(currentRace' ) BECOMES
  Number_Of_Selected' (currentRace') - 1
& Picked (cName, Race_Title(currentRace'))
  BECOMES FALSE
& Display ( scrNumber' ) BECOMES
  Update ( Display' ( scrNumber' ) ,
          cName, UnMarked )
& tempVoteRecord (currentRace') BECOMES
  tempVoteRecord' (currentRace')SET_DIFF
  { SETDEF C: Candidate (
    Candidate_Name ( C ) = cName) }
[...]
```

Another important transition to discuss is how we specify `Button_Push`, which is also an exported transition and, therefore, is called by the voter. Notice that each screen is associated with an integer value and, in some cases, with a particular name. To make the specification simple but without losing generality, we assume one race per screen. In reality, however, a screen can display more than one race. The voter and the poll worker interact with the screen while casting votes and administering the election (such as, loading a ballot prior to starting an election, initializing a voter's ballot, or dealing with abnormal situations). A voter navigates from one screen to another, by calling the transition

```
Transition Push_Button ( b: Button )
```

The transition has ten entry/exit pairs that correspond to the ten buttons defined previously. The entry and exit assertions that correspond to the `START` button are as follows:

```
ENTRY
  b = START
  & b ISIN Screen_Buttons ( curScreen )
  & scrName = START_SCREEN
  & curScreenNum = 0
  & ~Button_Pushed ( b )
  & Which_Phase = During_Voting
  & Terminal_Mode = voter_mode
```

```
& ~Signal_Enabled
& Which_Signal = NoSignal
```

The first five conjuncts specify conditions about the button and the current screen. They specify that the button that the voter pushed is `START`, the button is in the screen button list for the current screen, the current screen is `START_SCREEN`, the corresponding screen number equals zero, and the start button was not previously pushed. The next two conjuncts deal with election period and the status of the DRE terminal. The election phase must be during-voting and the terminal mode is voter-mode. The last two conjuncts of the entry assertion are used by the DRE to modulate the signaling information in order to alert the RTAL.

```
EXIT
  Button_Pushed ( b ) BECOMES TRUE
  & scrNumber = 1
  & currentRace = Which_Race ( scrNumber )
  & Screen_Buttons(curScreen ) = { NEXT }
  & Display ( scrNumber ) BECOMES
    Display_Contest(Race_Title(currentRace)
  ,Displayed_Candidates' ( currentRace )
  ,Screen_Buttons ( scrNumber ) )
  & voterNumber =voterNumber' + 1
  & Signal_Enabled
  & Which_Signal = Start_Signal
  & RTALMessage = VOTE_SESSION_STARTED
```

The exit assertion for the start case indicates that the voter has pushed the `START` button, the screen number is incremented by one, the current race is updated, the current screen displays the first race, the only button available to push is `NEXT`, and the number of voters who visited the poll is incremented by one. In addition, the DRE updates the value of the signaling variables and `RTALMessage` to be printed out on the paper tape.

The entry assertions for the rest of entry/exit pairs are more or less identical to the first five conjuncts of the start case except the button being pushed is different in each case (with additional conjuncts if applicable). The exit assertion, however, for each button press can be different depending on which button was pushed. Below, we discuss the exit assertions for the `NEXT` and `CONFIRM` button pushes.

The exit assertion of the “next button” indicates that the voter has pushed the `NEXT` button, that the `scrNumber` increments by one, which, in turn, is used by `Race_Title` to update the `currentRace` so that the corresponding candidates for the `currentRace` are displayed. If the voter has still more races to go, then the button choices in `Screen_Buttons` are `BACK` and `NEXT`. However, if the `currentRace` is the last contest, then the button choices available are `BACK` and `REVIEW`.

After the voter has completed all of his/her votes, the voter has to cast and confirm the choices. Once the voter touches the CAST button and confirms the vote by touching the CONFIRM button, the DRE updates the total tally in the exit assertion of confirm. Note that when the voter reaches the end of the ballot, they will be prompted to press the REVIEW button. When the REVIEW button is pressed the voter will be notified of any unvoted, or undervoted contests or if the ballot has been left blank. The voter has the option of reviewing their ballot and making any changes (by using the BACK button or by touching on the candidate name) before casting their ballot. In this paper we specified the change only by using the BACK button push until the voter reaches to the screen that contains the candidate. Pressing the CONFIRM button will cast the ballot.

```

/*exit assertion for 'confirm' button*/
EXIT
Button_Pushed ( b ) BECOMES TRUE
& FORALL C: Candidate, R: Race
( C ISIN Race_Candidates ( R )
& (IF
Picked' (Candidate_Name( C ),
Race_Title ( R ) )
THEN
TotalTallyCount ( C, R ) =
TotalTallyCount'( C, R ) + 1
ELSE
NOCHANGE (TotalTallyCount ( C, R ) )
FI ))
&(IF Min_Display' ( scrNumber' ) =
Display_Info ( Ballot_Not_Completed,
Screen_Buttons' ( scrNumber' ) )
THEN
NumberOfLogEntry = NumberOfLogEntry'+1
& EventLog ( NumberOfLogEntry )
BECOMES underVotedRaces'
& RTALMessage =
BALLOT_ACCEPTED_UNDERVOTE
ELSE
RTALMessage = BALLOT_ACCEPTED
& underVotedRaces = NoUnderVotedRace
FI)
& Signal_Enabled
& Which_Signal = Summary_Signal
& BallotBarcode =
BARCODE( voterNumber' )
& Terminal_Mode = sleep_mode
& scrNumber = - 1
& scrName = SETUP_SCREEN
& FORALL R: Race (
tempVoteRecord ( R ) = EMPTY )
& FORALL C: Candidate, R: Race (
Picked ( Candidate_Name ( C ),
Race_Title ( R ) ) = FALSE )

```

In addition to updating the total tally, the DRE keeps track of log data (such as undervoted races, if they ex-

ist, in underVotedRaces).

The DRE is also responsible for clearing the previous signal value and the button push, by performing the transitions Clear_Signal and Reset_Button_Push, respectively. We skip their detailed discussion.

When a voter flees (i.e., s/he leaves without confirming the vote), the iVotronic makes a chirping sound after ten minutes, which alerts a poll worker to correct the terminal. This is captured by the following transition:

```

TRANSITION Generate_Chirping_Sound
ENTRY [ TIME : G_C_Dur ]
( ( scrNumber >= 0 &
scrNumber <= Number_Of_Races + 2 )
& Now - 10 >= Change (scrNumber ) )
| Call (Make_Selection) -
Call[2](Make_Selection) >= 10 )
& Terminal_Mode = voter_mode
EXIT
Terminal_Mode = chirping

```

4.1.2 Modeling the RTAL Process

The RTAL is specified by an instance of type RTAL_Process. The RTAL prints vote actions exported by the DRE on a paper tape. The tape contains a list of voter records, where each voter record is a sequence of voter actions. These are declared using the following variables:

```

VARIABLE
Tape ( Pos_Integer ) : PrintValue,
tapePosition: Tape_Number,
VoteStartPosition ( Voter_Number ),
VoteEndPosition ( Voter_Number )
: Tape_Number,
RTAL_State: RTALState,
summaryPrinted: Boolean

```

The Tape represents the RTAL paper tape where the start information, vote selection, and summary information are continuously printed for each voter. After each print, the RTAL tapePosition is incremented appropriately. The variables RTAL_State and summaryPrinted, respectively, are used for keeping track of the current state of the RTAL and determining whether the summary information has been printed. This is to know when to scroll the tape forward by some amount in order to protect the secrecy of the previous voted ballot.

The variables VoteStartPosition and VoteEndPosition delineate the voter record on the paper tape.

```

TRANSITION Print_Selection
ENTRY
My_DRE.Plugged_In
& My_DRE.Signal_Enabled

```

```

& RTAL_State = Wait
& My_DRE.Which_Signal = Start_Signal
| My_DRE.Which_Signal = Vote_Signal
| My_DRE.Which_Signal = Summary_Signal

```

The first three conjuncts in the entry assertion specify that the RTAL has been plugged in to the DRE, that the DRE has sent a signal, and RTAL is waiting for the DRE signal to print. The fourth conjunct specifies what type of information the DRE is signaling to print.

```

EXIT
  RTAL_State = Printed
  & IF
    My_DRE.Which_Signal = Start_Signal
    | My_DRE.Which_Signal = Vote_Signal
  THEN
    tapePosition = tapePosition' + 1
    & CutLengthCounter =
      CutLengthCounter' + 1
    & ( IF
      My_DRE.Which_Signal = Start_Signal
      THEN /*start printing*/
        voterNumber = voterNumber' + 1
        & Tape ( tapePosition ) BECOMES
        Make_Print_Info(My_DRE.RTALMessage)
      ELSE /*vote entry printing*/
        Tape ( tapePosition ) BECOMES
        Make_Print_VoteEntry (
          My_DRE.pickedName,
          My_DRE.currentRace,
          My_DRE.pickedValue )
      FI)
  ELSE
    tapePosition =
      tapePosition' + 3
    & CutLengthCounter =
      CutLengthCounter'+3
    & Tape ( tapePosition - 2 ) =
      Make_Print_Info (
        My_DRE.RTALMessage )
    & Tape ( tapePosition - 1 ) =
      Make_Print_Undervote (
        My_DRE.underVotedRaces )
    & Tape ( tapePosition ) =
      Make_Print_BallotBarcode (
        My_DRE.BallotBarcode )
    & FORALL i: Tape_Number
      ( i ~= tapePosition
        & i ~= tapePosition - 1
        & i ~= tapePosition - 2
        -> NOCHANGE ( Tape ( i ) ) )
    & VoteStartPosition ( voterNumber )
      BECOMES tapePosition -
        CutLengthCounter +1
    & VoteEndPosition ( voterNumber )
      BECOMES tapePosition
    & summaryPrinted

```

FI

After the transition is fired, depending on what signaling mode has been received by the RTAL, the corresponding entry is printed.

After printing the summary information, the printer is also scrolled forward, which is performing by the transition

```

TRANSITION Scroll_Forward

```

4.1.3 Modeling the PEB Process

The PEB device is specified by an instance of type PEB_Process. As mentioned earlier, the PEB devices—in addition to their being used to load the ballot data into iVotronic terminals prior to starting the election and to initialize a ballot when a voter comes during election—are used to transfer election specific data between Election Central and poll locations, which is indicated by the variables

```

VARIABLE
  Candidates_Of_Race ( Race )
    : Candidates,
  tabulatedData (Candidate, Race, DRE_ID)
    : Non_Negative,
  copyOfBallotImages ( DRE_ID )
    : StoreBallot,

```

Candidates_Of_Race stores the ballot data to be loaded into each iVotronic terminal for the current election prior to starting a voting session. tabulatedData stores the election results as it was stored within the DRE, whereas copyOfBallotImages contains the ballot images as used for the election. The reader should note that according to the ES&S voting process specification, the iVotronic terminal authenticates each PEB by its four digit EQC code. While all PEBs are internally identical in construction, they are discernible from one another by the read only information burned in the PIC: their serial number, and more importantly by their PEB kind, namely either “master” or “supervisor”. In this specification, we only use PEB kinds to distinguish PEBs.

```

VARIABLE
  Secret_EQC : Digit_List,
  Kind : PEBKind,

```

The most important aspect to specify about the PEB process is that after the terminal is closed, the poll worker uses the master PEB to collect and store the tabulated data and copies of the “images” of the ballots. The following transition is responsible for doing that.

```

TRANSITION Download_Results ( D: DRE_ID )
ENTRY [ TIME : P_D_Dur ]
  D.enabled_PEB_Download
  & D.PEB_Inserted
  & D.Which_PEB_Inserted = Self
  & Kind = Master
  & ~ResultDownload_Completed ( D )
  & copyOfBallotImages ( D ) =
    NoBallotImage
EXIT
  ResultDownload_Completed ( D )
    BECOMES TRUE
  & FORALL C: Candidate, R: Race
    ( C ISIN D.Race_Candidates ( R )
      -> tabulatedData ( C, R, D ) =
        D.TotalTallyCount ( C, R ) )
  & copyOfBallotImages ( D ) BECOMES
    Download_BallotImage (
      { SETDEF Pair: Race_Candidates_Pair(
        EXISTS R: Race
          ( Pair [ Contest ] = R
            & Pair [ Nominees ] =
              D.Race_Candidates ( R ) ) ) } )
  & ResultDownload_Completed ( D )
    BECOMES TRUE

```

4.1.4 Modeling the CF Card Process

The Compact Flash Card is specified by an instance of type `CFCard_Process`. An audit file is saved automatically to the card (see, the `Download_AuditData` transition below) when the polls are closed. From a formal specification point of view, however, we are only interested in the audit log file, which contains the under-voted races and the number of fleeing voters, indicated by the following variables:

```

VARIABLE
  EventLog (Pos_Integer ): Races,
  numOfFleeingVoters,
  visitedNumberOfVoters : Non_Negative,
  ADDownload_Completed: Boolean

```

There is one Compact Flash Card per DRE machine used in the election. This card is uniquely identified by its serial number:

```

VARIABLE
  SerialNumber: Digit_List

```

As mentioned previously, upon closing the terminal while the master PEB is inserted, the DRE automatically enables the CF card to save audit data. This activity is modeled by the following transition

```

TRANSITION Download_AuditData
ENTRY [ TIME : CF_D_Dur ]
  My_DRE.CFCard_Installed

```

```

  & My_DRE.Which_CFCard_Installed = Self
  & My_DRE.enabled_CFCard_Download
  & ( visitedNumberOfVoters = 0
    & numOfFleeingVoters = 0
    & FORALL n: Non_Negative
      ( EventLog ( n ) = EMPTY ))
  & ~ADDownload_Completed
EXIT
  FORALL n: Pos_Integer
    ( n <= My_DRE.NumberOfLogEntry
      -> EventLog ( n ) =
        My_DRE.EventLog ( n ) )
  & numOfFleeingVoters =
    My_DRE.numberofFleeingVoters
  & visitedNumberOfVoters =
    My_DRE.voterNumber
  & ADDownload_Completed

```

In order to fire this transition, the same CF card must be present and there is no audit log collected so far. If this is the case, then for each DRE log entry n `EventLog(n)` is updated, the number of fleeing voters and the total number of voters who came to cast their votes (including the fleeing voters) are copied, and the `ADDownload_Completed` is set to `TRUE`, which notifies a poll worker to remove the inserted PEB and CF card after a reasonable amount of time has elapsed (see, the `Remove_PEB` transition in Section 4.1.1).

4.2 Critical Requirement Specifications

Once we model the main components of the system and its evolution, we need to model what critical requirements the system should meet given the assumptions about the behavior of the system and the external environment that interacts with the system. There are a number of behaviors we need to specify about the external environment that the e-voting system relies on. The behavior of the people (voters, poll workers, and election officials) who interact with the system is outside the DRE system, but it influences how the system operates. In fact, the DRE cannot control the behavior of the voter when s/he interacts with the screen. For example, if the voter touches the candidate name faster than DRE can process the touches, the normal functioning of the e-voting system may be disrupted. In addition, the procedures that control the voting process are completely outside of the e-voting system, but they are equally important to carry out correct and secure elections. Therefore, we need to express these concerns in order to guarantee the critical requirements that the system should meet.

In what follows we discuss some of the assumptions we made for the ES&S specification presented in this paper. We assume that a voter will pause for some amount of time between subsequent screen interactions. This at

least guarantees that the selection or button push is processed by the DRE. Assumptions about poll workers' operations on election day also need to be made. For instance, removing an inserted PEB during the ballot retrieving process is dangerous, therefore, the poll worker should wait until the DRE process notifies him or her to remove the PEB.

The local environment clause for the process type `DRE_Process` is

```
ENVIRONMENT
FORALL t: time
  ( Call [ 2 ] ( Make_Selection, t )
    -> Call ( Make_Selection ) - t
      > min_pause )
& FORALL t: time
  ( Call [ 2 ] ( Push_Button, t )
    -> Call ( Push_Button ) - t
      > min_pause )
& FORALL t: Time
  ( Call ( Insert_PEB, t )
    -> ~past ( PEB_Inserted, t ) )
& FORALL t: Time
  ( Call ( Remove_PEB, t )
    -> past ( PEB_Inserted, t ) )
& FORALL t: Time, b: Button
  ( Call ( Push_Button ( b ) , t )
    -> ~past ( Button_Pushed(b) , t ) )
& EXISTS t: Time
  ( Call ( Turn_DRE_On, t )
    -> ~past ( MachineTurnedOn, t ) )
& EXISTS t: Time
  ( Call ( Turn_DRE_Off, t )
    -> past ( MachineTurnedOn, t ) )
```

This states that there must be a minimum pause between two subsequent selections and button pushes. It also specifies that a poll worker will not attempt to insert/delete a PEB or turn the DRE on/off when the iVotronic is not in the appropriate state.

Similarly, the clause

```
ENVIRONMENT
FORALL t: Time
  ( Call [ 2 ] ( Print_Vote_Record, t )
    -> Call ( Print_Vote_Record ) - t
      >= print_time )
```

specifies a local environment for the `RTAL_Process`, which states that `print_time` is the time elapsed between two subsequent print calls.

The integrity of the election results is expressed by the following global invariant (*G-Req2*):

```
EXISTS p: PEB_Number
  ( the_PEB [p].Kind = Master
    & FORALL d: DRE_Number
```

```
  ( the_PEB[p].ResultDownload_Completed
    ( the_DRE [d].Self )
    & the_DRE [d].Which_Phase = Post_Voting
    & the_DRE [d].DRE_State = Closed
    & FORALL C: Candidate, R: Race
      (C ISIN
        the_DRE [d] .Race_Candidates ( R )
        -> the_PEB [p].tabulatedData
          (C,R, the_DRE [d].Self ) =
          the_DRE [d].TotalTallyCount (C,R))))
&
EXISTS p: PEB_Number
  (the_PEB [ p ] .Kind = Master
    & FORALL d: DRE_Number, rt: RTAL_Number
      (the_RTAL [rt] =
        Plugged_In_RTAL ( the_DRE [ d ] )
        & the_PEB [p].ResultDownload_Completed
          (the_DRE[d].Self )
        & the_DRE [d].Which_Phase = Post_Voting
        & the_DRE [d].DRE_State = Closed
        & FORALL C: Candidate, R: Race
          (C ISIN
            the_DRE [ d ] .Race_Candidates ( R )
            -> the_PEB[p].tabulatedData
              CountCanceled(C,R,the_RTAL[rt])))
```

the first conjunct of the invariant says that there exists a PEB `p`, such that for every DRE `d` in the precinct, if `p` is the master PEB used in `d`'s terminal to download the election results after the `d` terminal is closed and the election has ended (i.e., `Post_Voting` phase), then the election results for each candidate `C` who ran for race `R` stored in `p` is exactly equal to the total tally counted on `d`'s terminal for candidate `C`. Similarly, the second conjunct specifies that for every RTAL printer `rt` and DRE `d` in the precinct, if `rt` is the printer used by `d` during the voting period, then the election result for each candidate `C` who ran for race `R` stored in `p` is the difference between the total number of selected and the total number of canceled votes printed on `rt`.

The number of rejected ballots printed on the RTAL must match with the number of fleeing voters in the PEB (a fleeing voter is a voter who does not complete the voting procedure). Moreover, the number of voters who visited the polling place where they registered to cast their votes must match with the number of voter records printed on the RTAL, whether the voter confirmed his/her ballot or rejected it. These facts are expressed by the following two global invariants (*G-Req3* & *4*):

```
FORALL cf:CFCard_Number,rt: RTAL_Number
  ( the_CFCard[cf].ADDDownload_Completed
    & EXISTS d: DRE_ID
      (the_CFCard[cf]
        = Installed_CFCard ( d )
        & the_RTAL[rt]
          = Plugged_In_RTAL ( d )
          & d.Which_Phase=Post_Voting
```

```

    & d.DRE_State=Closed)
    -> the_CFCard[cf].numOfFleeingVoters
        = CountRejected(the_RTAL[rt]))
&
FORALL cf: CFCard_Number, rt: RTAL_Number
( the_CFCard [cf].ADDdownload_Completed
& EXISTS d: DRE_Number
( the_CFCard[cf] =
  Installed_CFCard ( d )
& the_RTAL[rt] =
  Plugged_In_RTAL ( d )
& d.Which_Phase =
  Post_Voting
& d.DRE_State = Closed )
-> the_CFCard [cf].voterNumber
    = the_RTAL[rt].voterNumber )

```

We mentioned that once a Compact Flash Card is installed into the DRE terminal prior to start election, it must be present throughout the voting process. This fact is expressed by the local invariant of the DRE_Process (*D-Req1*):

```

Change ( CFCard_Installed, Now )
& ~CFCard_Installed
-> EXISTS c: CFCard_ID (
  past(Which_CFCard_Installed, 0) = c
& FORALL t: Time (
  t >= 0
& t < Now
& Stored_SerialNumber =
  past ( Stored_SerialNumber, 0)
& past (CFCard_Installed, t) ) )

```

Another local invariant of DRE_Process which indicates the fact that at any time the number of selections for a given race R can never be more than the allowed number of choices (*D-Req8*):

```

FORALL R: Race (
  Change(Number_Of_Selected( R ), Now)
& Number_Of_Selected ( R )
  ~ = Number_Of_Selected' ( R )
-> Number_Of_Selected ( R )
  <= Max_Choice_Per_Race ( R ) )

```

The constraint (*D-Req10*):

```

FORALL C: Candidate, R: Race (
  EXISTS b: Button (
    b = CONFIRM
& ~Button_Pushed' ( b )
& Button_Pushed ( b ) )
& TotalTallyCount' ( C, R )
  ~ = TotalTallyCount ( C, R )
-> TotalTallyCount ( C, R ) =
  TotalTallyCount' ( C, R ) + 1
  & Picked' ( Candidate_Name ( C ),
    Race_Title( R ) )

```

```

& ~Picked( Candidate_Name ( C ),
  Race_Title( R ) ) ) )

```

states that whenever the voter confirms, the total tally recorded in DRE must be updated for the candidates whose picked value is true.

In a similar manner, the fact that when a voter selects or cancels a candidate C for a given race R, the DRE screen must be updated accordingly is expressed by the following two conjuncts (*D-Req5*):

```

FORALL C: Candidate, R: Race(
  Fill ( Picked' ( Candidate_Name ( C ),
    Race_Title ( R ) ) ) = UnMarked
& Fill ( Picked ( Candidate_Name ( C ),
  Race_Title ( R ) ) ) = Marked
& Display ( scrNumber' ) ~ =
  Display' ( scrNumber' )
-> Display ( scrNumber' ) =
  Update ( Display' ( scrNumber' ) ,
    Candidate_Name ( C ) , Marked ) )
& FORALL C: Candidate, R: Race
( Fill ( Picked ( Candidate_Name ( C ),
  Race_Title ( R ) ) ) = UnMarked
& Fill ( Picked' ( Candidate_Name ( C ),
  Race_Title ( R ) ) ) = Marked
& Display ( scrNumber' ) ~ =
  Display' ( scrNumber' )
-> Display ( scrNumber' ) =
  Update ( Display' ( scrNumber' ) ,
    Candidate_Name( C ) , UnMarked ) )

```

The fact that a DRE is chirping indicates that at least ten minutes have passed since the last ballot activity. This is expressed by the following local schedule requirement of the DRE_Process (*D-Req11*):

```

Change ( Terminal_Mode, Now )
& Terminal_Mode = chirping
-> ( Now - Change ( scrNumber ) >= 10
  & EXISTS t: Time (
    t <= Now
& t > Change[2](Terminal_Mode)
& past(Terminal_Mode,t)
  = voter_mode))
| Call ( Make_Selection ) -
  Call [2] ( Make_Selection ) >= 10

```

which says, the mode of the terminal is set to *chirping* if there is no user input to the DRE within ten minutes since last screen change or the last call to the exported transition Make_Selection by the voter.

We mentioned that the RTAL must print the corresponding voter activity on the tape. This requirement must be expressed as a scheduling requirement because the printing activity depends on the signal information sent by the DRE_Process through the Signal_Enabled

variable. The schedule clause for the `RTAL_Process` consists of four conjuncts, each corresponding to a scheduling requirement. Below, we present one of them.

The vote entry—a tuple that consists of a candidate, a race and the value of the selection— will be printed on the `RTAL` tape one tape position below the previous print if the `DRE` has enabled the signal, made available the information to print, and enough time has elapsed for the choice to be printed (*R-Req2*):

```
My_DRE.Signal_Enabled
& past ( My_DRE.Which_Signal,
         Change( My_DRE.Signal_Enabled ) )
      = Vote_Signal
& Now - Change ( My_DRE.Signal_Enabled )
          > Max_Print_Time
-> EXISTS t: Time (
  t > Change ( My_DRE.Signal_Enabled )
  & t <= Now
  & Change ( tapePosition, t )
  & past ( tapePosition, t ) =
    past ( tapePosition, Now -
          Change(My_DRE.Signal_Enabled)) + 1
  & Tape ( tapePosition ) =
    Make_Print_VoteEntry (
      My_DRE.pickedName,
      My_DRE.currentRace,
      My_DRE.pickedValue ) ) )
```

4.3 Verification Using PVS

The specification presented in the previous section was constructed and type-checked using the `ASTRAL SDE` [15]. We validated the specification and generated the corresponding proof-obligations for the critical requirements. Moreover, the specification was automatically translated into its `PVS` [23] counterpart using the `ASTRAL SDE`, which enabled the specification to be passed to the `PVS` theorem-prover for verification.

Before invoking the theorem prover, the `ASTRAL` split engine was used to split the `ASTRAL` specification into collections of simpler properties that infer the whole clause so that the proof of each property can be tackled separately. For instance, one of the conjuncts that specify the requirement *D-Req1* is split into four smaller properties: one untimed invariant and three safety properties.

Table 1 shows the number of invariants, schedules, and constraints for each of the four processes and the global invariants. It also shows the number after they are split by the `ASLAN SDE`.

Using the `PVS` interactive theorem prover and the techniques discussed in [16] we have proved many critical requirements for the system (mostly local invariants and constraints). For example, for the split of the *D-Req1* property discussed above, the first split was trivially proved using the *try-untimed* strategy and the re-

Table 1: Number of proof obligations.

	Proof Obligations	After Splitting
	Invar, Sched, Constr	Invar, Sched, Constr
DRE	4, 1, 6	10, 2, 9
RTAL	1, 3, 1	1, 3, 1
PEB	1, 1, 0	2, 1, 0
CFCard	0, 1, 0	0, 2, 0
Global	6, 0, 0	9, 0, 0

maining three splits were proved using the `Safety Property Proof` technique.

At this time, we have not yet completed all of the proofs. More specifically we have proved 6 of the 22 invariants, 2 of the 8 schedules, and 4 of the 10 constraints. We expect that the other global and local properties can be proved using the same or similar proof techniques and strategies. We were initially slowed down due to changes in the `PVS` theorem prover. We have now updated the `ASTRAL` environment to be consistent with the latest version of `PVS`, and we should have all of the properties proved in the near future.

5 Related Work

Scientific literature on e-voting is wide and multi-disciplinary. Sticking to the topic of this paper, we organize previous work in three different areas: understanding the risks posed by the introduction of e-voting systems in the polling stations; assessing existing systems; designing better e-voting systems using formal techniques.

With respect to the first area, work in the past has focused on understanding what changes could be introduced in the “traditional” voting procedures to allow a secure transition to electronic elections. For instance, [33, 32] discuss risks and difficulties related to the introduction of e-voting, [29, 25] suggest possible improvements to existing procedures, and [34, 31, 30] introduce techniques to formally analyze what security breaches may be derived by executing the procedures in the wrong way. Our work complements those presented above, by helping, e.g., understand how to allocate requirements between procedures and systems to provide more secure electronic elections.

With respect to the second area, assessing existing systems some e-voting systems currently deployed in elections have recently undergone a thorough and independent scrutiny to evaluate their quality. See, for instance,

[1, 2, 4, 26], where the authors highlight some serious design and implementation flaws, which could be exploited to compromise elections. The papers also suggest a drastic change in the way in which electronic voting systems are designed, developed, and tested.

With respect to the third area, several works describe the usage of formal method techniques to specify, model, analyze, and assess complex and safety-critical systems (see, for instance, [10, 9, 19, 5] and [18, 11, 6, 24]). The application of formal methods in the e-voting area, however, has been, so far, limited. We mention [12, 17], that present the formal specification and verification of an electronic voting protocol using Pi calculus and [27], where the authors present a tool, FSMC+ that has been used to specify and verify the control logic of an e-voting machine. Our work builds upon and improves those presented above by widening the scope of the analysis and by taking into account aspects related to the procedures and interaction of the system with its environment.

6 Conclusion and Future Work

In spite of the potential advantages e-voting might bring to the polling station, such as improved turn out, accessibility for impaired people, and improved accuracy and speed, its adoption in various countries has been slow and/or the cause of great debates and controversies.

One of the reasons is that e-voting machines are complex real-time embedded systems required to operate in a (possibly) hostile environment. Another and more relevant reason is poor design and poor implementation of (some of) the systems currently deployed for elections in the US and in other countries, as different studies have reported and demonstrated. Such weaknesses expose e-voting systems, and consequently elections, to threats and attacks, whose effects vary from a “denial of service” (e.g., stopping the election in a polling station by sabotaging some e-voting machines) to alteration of the results (e.g., by successfully changing votes in some key precincts).

In this paper we have shown how formal verification techniques can be used to model and reason about the security of e-voting systems. This paper does not consist of a novel specification technique nor a novel voting system. Instead, it presents the formal specification of the voting process for the ES&S system. The specification was constructed and type-checked using the ASTRAL SDE [15]. We validated the specification using the ASTRAL tool, and the specification was passed to the PVS [23] analysis tool.

So far we managed to formally verify that the specification satisfies many of the critical requirements that we discussed in this paper, mostly the local invariants and constraints. The proofs were achieved by following the

techniques presented in [16]. For instance, we applied the *try-untimed* and *try-untimed-con* proof strategies to prove some of the local invariants and constraints of the system. In general, the proof is carried out first by splitting the critical requirements and applying the appropriate proof strategies developed to support the ASTRAL analysis (see [16]).

Currently we are verifying the remaining requirements as they are specified in the global and local invariants, schedules, and constraints. We also plan to formalize some of the attacks on the ES&S system that were discussed in [2, 4, 21]. We expect that by analyzing these attacks we will find missing critical requirements for our specifications or assumptions that were not met.

Even though choosing whether to adopt e-voting or guaranteeing secure (electronic) elections cannot be reduced simply to the issue of building more secure and robust e-voting machines, achieving such a goal is a fundamental step. Formal verification is one of the techniques that can help us improve the correct and secure development of e-voting systems.

References

- [1] Nirwan Ansari, Pitipatana Sakarindr, Ehsan Haghani, Chao Zhang, Aridaman K. Jain, and Yun Q. Shi. Evaluating electronic voting systems equipped with voter-verified paper records. *IEEE Security and Privacy*, 6(3):30–39, 2008.
- [2] Adam Aviv, Pavol Cerny, Sandy Clark, Micah Sherr Eric Cronin, Gaurav Shah, and Matt Blaze. Security Evaluation of ES&S Voting Machines and Election Management System. In *In Proc. of the USENIX/ACCURATE Electronic Voting Technology Workshop*, 2008.
- [3] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.
- [4] Davide Balzarotti, Greg Banks, Marco Cova, Viktoria Felmetzger, Richard Kemmerer, William Robertson, Fredrik Valeur, and Giovanni Vigna. Are Your Votes Really Counted? Testing the Security of Real-world Electronic Voting Systems. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 237–248, 2008.
- [5] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. NuSMV 2: An Open Source Tool for

- Symbolic Model Checking. In *Proc. of International Conference on Computer-Aided Verification*, 2002.
- [6] Chiara Braghin, Natasha Sharygina, and Katerina Barone-Adesi. Automated Verification of Security Policies in Mobile Code. In Jim Davies and Jeremy Gibbons, editors, *IFM*, volume 4591 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2007.
- [7] J W. Bryans, B Littlewood, P Y. A. Ryan, and L Strigini. E-voting: Dependability Requirements and Design for Dependability. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 988–995, Washington, DC, USA, 2006. IEEE Computer Society.
- [8] Alberto Coen-Porisini, Carlo Ghezzi, and Richard A. Kemmerer. Specification of Real-time Systems Using ASTRAL. *IEEE Trans. Softw. Eng.*, 23(9):572–598, 1997.
- [9] R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed model checking of security protocols. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 23–32, New York, NY, USA, 2004. ACM.
- [10] Zhe Dang and Richard A. Kemmerer. A Symbolic Model Checker for Testing ASTRAL Real-Time Specifications. *Real-Time Computing Systems and Applications, International Workshop on*, 0:174, 1999.
- [11] Zhe Dang and Richard A. Kemmerer. Using the ASTRAL model checker to analyze mobile IP. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 132–141, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
- [12] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying Privacy-type Properties of Electronic Voting Protocols. Research Report LSV-08-01, Laboratoire Spécification et Vérification, ENS Cachan, France, January 2008. 55 pages.
- [13] Election Systems & Software Inc. (ES&S). iVotronic™ Voting System. Version 9.1.x Election Day Operations Checklist, Revision Date: January 2007.
- [14] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an Electronic Voting System. *Security and Privacy, IEEE Symposium on*, 0:27, 2004.
- [15] Paul Z. Kolano. ASTRAL Software Development Environment User’s Manual. Technical report, Santa Barbara, CA, USA, 1997.
- [16] P.Z. Kolano. *Tools and Techniques for the Design and Systematic Analysis of Real-Time Systems*. PhD thesis, University of California, Santa Barbara, 1999.
- [17] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In Mooly Sagiv, editor, *Programming Languages and Systems — Proceedings of the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200, Edinburgh, U.K., April 2005. Springer.
- [18] Michael Lowry and Daniel Dvorak. Analytic Verification of Flight Software. *IEEE Intelligent Systems*, 13(5):45–49, 1998.
- [19] Nicolas Markey and Philippe Schnoebelen. Tsmv: A symbolic model checker for quantitative analysis of systems. *Quantitative Evaluation of Systems, International Conference on*, 0:330–331, 2004.
- [20] Matt Bishop and David Wagner. Risks of e-voting. *Commun. ACM*, 50(11):120–120, 2007.
- [21] P. McDaniel, M. Blaze, and G. Vigna. EVEREST: Evaluation and Validation of Election-Related Equipment, Standards and Testing. Ohio Secretary of State’s EVEREST Project Report, December 2007.
- [22] Anne-Marie Oostveen and Peter Van den Besselaar. Security as Belief User’s Perceptions on the Security of E-Voting Systems. In *Electronic Voting in Europe*, pages 73–82, 2004.
- [23] S. Owre, N. Shankar, and J. M. Rushby. *The PVS Specification Language*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.
- [24] Gordon J. Pace, Cristian Prisacariu, and Gerardo Schneider. Model checking contracts - a case study. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2007.
- [25] Alexander Prosser, Robert Kofler, Robert Krimmer, and Martin Karl Unger. Security assets in e-voting. In Alexander Prosser and Robert Krimmer, editors, *Electronic Voting in Europe*, volume 47 of *LNI*, pages 171–180. GI, 2004.

- [26] Ryan Gardner and Sujata Garera and Aviel D. Rubin. On the Difficulty of Validating Voting Machine Software with Software. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology*, pages 11–11, Berkeley, CA, USA, 2007. USENIX Association.
- [27] Roberto Tiella, Adolfo Villafiorita, and Silvia Tomasi. Fsmc+, a tool for the generation of java code from statecharts. In *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, pages 93–102, New York, NY, USA, 2007. ACM.
- [28] Roland Traummüller, editor. *Electronic Government: Third International Conference, EGOV 2004, Zaragoza, Spain, August 30 - September 3, 2004, Proceedings*, volume 3183 of *Lecture Notes in Computer Science*. Springer, 2004.
- [29] Melanie Volkamer and Robert Krimmer. Independent audits of remote electronic voting developing a common criteria protection profile. In *Proceedings der EDEM 2007 Elektronische Demokratie in sterreich*, 2007.
- [30] Komminist Weldemariam and Adolfo Villafiorita. Modeling and Analysis of Procedural Security in (e)Voting: The Trentino's Approach and Experiences. In David L. Dill and Tadayoshi Kohno, editors, *EVT*. USENIX Association, 2008.
- [31] Komminist Weldemariam, Adolfo Villafiorita, and Andrea Mattioli. Assessing Procedural Risks and Threats in e-Voting: Challenges and an Approach. In Ammar Alkassar and Melanie Volkamer, editors, *VOTE-ID*, volume 4896 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2007.
- [32] Alexandros Xenakis and Ann Macintosh. G2g collaboration to support the deployment of e-voting in the uk: A discussion paper. In Traummüller [28], pages 240–245.
- [33] Alexandros Xenakis and Ann Macintosh. Levels of difficulty in introducing e-voting. In Traummüller [28], pages 116–121.
- [34] Alexandros Xenakis and Ann Macintosh. Procedural Security Analysis of Electronic Voting. In *ICEC '04: Proceedings of the 6th international conference on Electronic Commerce*, pages 541–546, New York, NY, USA, 2004. ACM Press.
- [35] Dianxiang Xu and Kendall Nygard. A Threat-Driven Approach to Modeling and Verifying Secure Software. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 342–346, New York, NY, USA, 2005. ACM Press.