Managing Requirements for e-voting Systems: Issues and Approaches Motivated by a Case Study

Komminist Weldemariam, Andrea Mattioli, Adolfo Villafiorita  
eDemocracy Group  
Center For Information Technology, FBK-IRST  
Trento, Italy  
Email: (sisal,amattioli,adolfo)@fbk.eu

Komminist Weldemariam  
Dept. of Information Eng. and Computer Science  
University of Trento  
Trento, Italy  
Email: weldemar@disi.unitn.it

Abstract—This paper discusses our approach and experiences on structuring and maintaining requirements for an e-voting system we have built and deployed for elections. Issues related to integrating laws and recommendation for e-voting systems, managing different elections and configurations, supporting a spiral development, yielded problems and approaches to help maintain integrity of requirements and a coherent view of the system. Moreover, the relationship between requirements and system architecture is based on finite state machines, that bridge the gap between the laws and the actual behavior of the machine.

Keywords—Architecture, Electronic Voting System, Finite State Machine, Requirements, Requirement Structure and Management.

I. INTRODUCTION

Various publicly available standards, principles, and recommendations in e-voting —e.g., the FEC [1], [2], EU Venice Commission [3], and McGaley reports [4], [5], [6], address what an e-voting system should reliably do. From requirement engineering point of view, however, these documents often specify the what do concerns. An example from the FEC (also in [7]): voting system always gives the voter a chance to review their ballot and correct any mistakes they discover before casting.

This is natural, because these documents have to encompass a large range of election practices and systems. However, when it comes down to building an e-voting system, they lack in providing those lower-level requirements that are essential to support the actual development activities.

This paper reports on our experiences in structuring requirements for the implementation and formal verification of an e-voting system, which we deployed for different kinds of elections in different hardware configurations. We achieved these goals by providing a hierarchical top-down structuring of the requirements. This allowed us to provide traceability from higher level principles up to the architectural level, generate the specification of a generic layer based on finite state machines on which we applied formal verification techniques, and clearly separate election and configuration specific requirements.

Although the work is motivated by a specific implementation (the ProVotE e-voting system, see [8], [9]), we believe that the approach and the methodology we devised are fairly general to be used for other e-voting systems and, possibly, to provide a road-map —rough and draft as it might be— for bridging the gap between higher level principles and lower level system specifications.

The paper is structured as follows. In the next section we provide a general description on e-voting machine we built. Section III states the problem statement for this work. In Section IV, we describe the methodology and our experience on structuring e-voting requirements. Section V discusses the mapping of requirements into a concrete implementation. We mention related work, and draw our conclusions and possible improvement in Sections VI and VII.

II. BACKGROUND: THE PROVOTE MACHINE

This section provides background information for this paper on the ProVotE project (see, [8], [10], [11], [12], [13], for previously published works). The reader familiar with DRE-VVPAT or with the ProVotE project may skip this section.

The design of the ProVotE system was aimed to sketch the machine as closer as the paper-based election. From a very high-level perspective, the ProVotE system is based on various sub-systems, which address two main process flows: the electronic voting and verification flows (Figure 1).

The first chain includes the electronic devices responsible for automation of the voting procedures. The Configurator is responsible for uploading the candidates and encryption keys (used to verify if the results are coming from legitimate voting machine) into the e-voting system. The Voting Machine is responsible for casting votes and performing administrative operations during the election (e.g., testing the voting machine, tabulation of data, etc). The Vote Tally System is responsible for aggregating collected data and publishing provisional results.

The second chain shows the verification flow that are used to verify electronically produced data. Its main sub-systems...
are: Paper Ballot Counter—a barcode recounting system to read barcodes printed on the ballots, and Log analyzer—an application employed, after the election, to generate reports about the use of a voting machine based on its log information (e.g., how many times the machine has been started, average voting time).

The ProVotE voting machine is a direct recording electronics with voter verified printed audit trail (DRE-VVPAT, as suggested elsewhere [14]). Its requirements are mainly influenced by laws and procedures, operational modes, and interactions among systems and their components.

Laws and procedures have a significant influence on the e-voting machine requirements that define the actual voting process. For example, the law states that the voter who decided to abandon the voting process in any stage has to be marked in the voter’s registry. This was translated to a requirement that corresponds to the specification of the current state of the voter while s/he is voting, without disclosing voter’s privacy. This results in to device a mechanism for the poll workers to check if the voter has completed voting process correctly or withdrew from voting.

Requirements that are related to the interaction with other systems include the possibility to deal with data for different elections, to print a barcode compatible with Paper Ballot Counter, to log events compatible with the Log Analyzer, and so on. Moreover, the operational modes —i.e., administration and voting modes, led to requirements to isolate administrative functions from voting functions.

Some characteristics of the resulting machine, such as the screen size, printed audit trail, signaling system (outside the polling booth) to display the status of the machine to poll workers, the software and OS—respectively, Java and Linux based, make the machine quite unique.

III. PROBLEM STATEMENT

The main challenges we found in eliciting and maintaining requirements for the e-voting machine are:

1) **Information Sources.** Various documents (see section IV-A) provide high-level principles and recommendations for e-voting machines. However, when managing the transition from paper-based to electronic elections, such information has to comply with acts regulating elections. Integrating and maintaining traceability of the actual machine requirements from these principles becomes an important aspect to guarantee effective management of the requirements.

2) **Bridging the gap between requirements and architecture.** The project has been organized using an approach strongly inspired by the spiral development process, to pace development with electoral events. However, this required to define a simple and clear strategy for managing requirements across the different phases of the project and thus, to help ensuring integrity between requirements and system.

3) **Software Configuration.** The variation in ballot features, voting options, counting algorithms, etc — required to use the system for different elections (e.g., municipality, referendum, provincial, councils, and unions representatives)— increases the complexity of managing configurations and recognizing logical sections which allow the program to be modularized.

4) **Hardware Configuration.** Different electoral events and different electoral offices require customized configuration of the machines. On top of that, the natural evolution of the system made us to experiment with and introduce new hardware components (such as, PC, BIOS, external LCD display, Printers). Requirements had, therefore, to allow for a flexible management of configuration-specific requirements, such as those deriving from the usage of different hardware components.

5) **Integrations with the manual electoral procedures.** An important legal and usability concern is the compatibility of voting machines with the current electoral laws and processes. In fact, in order to provide a smooth transition from paper-based voting to e-voting, from the voters point of view the process should be as similar as possible to the manual one and poll workers should be able to recognize procedural steps they are familiar with.

In the next few sections, we describe how we partially addressed the challenges mentioned above by a carefully planned strategy for the representation and management of requirements.
IV. STRUCTURING AND MAINTAINING REQUIREMENTS

A. Requirements Structuring

Figure 2 shows an abstract view of the approach we took to elicitate and structure requirements for the ProVotE system.

The upper part of the picture lists the different sources of information that we evaluated to understand the election processes, identify constraints, and distinguish both common and event specific requirements. In the figure, the grey arrows highlight how the documents contributed to the definition of fine-grained requirements. In the rest of the picture, whereas, the decomposition from high-level to low-level requirements and the logical dependencies among them are shown.

In what follows we discuss the details of each layer and how we moved from principles to common, till event specific requirements.

1) Sources of Information: These were mainly collected from government and public documents. The Italian laws and procedures provide the specification of the election process mainly for paper-based elections (see, e.g., for the discussion in [8], [9], [11], [13]). The interviews with domain experts were also of fundamental importance to limit the legal boundary of the project and assess the voting procedures. In particular, the law precisely describes the paper ballot layout (which had to be represented electronically on the screen) and counting algorithms for each type of election. The procedures contributed to the definition of a general (election independent) finite state machine compatible with the law.

Various international suggestions (e.g., the McGaley reports [5], [6]), the European Union (EU) Venice Commission recommendations [3], and the U.S. Federal Election Commission (FEC) Voting Systems Standard (VSS) [1], [2], were mainly used to obtain a set of principles compliant with the local law. The FEC-VSS provides details about the standards to be used for performance and tests of the voting machines. It also describes non-functional requirements (e.g., audits log features) and specifications for various hardware components.

2) Principles: Principles are high-level (but not operational) requirements, which define the automation boundaries and system goals. These helped us to understand what system properties can be reasonably implemented (and possibly verified) and which responsibilities should be delegated to new procedures.

Examples may help: The guiding principles that the voter’s intent should be captured in an unambiguous manner and that election results should be (manually) reproducible. These facts were translated into machine features and procedural aspects. Namely, the machine should allow the voter to review his/her choices on the screen and on the VVPAT printer before casting the ballot (machine). The voter has to explicitly confirm the vote (procedure) and finally the machine has to mark the paper ballot confirmed or rejected before dropping it into a sealed ballot box. The requirements how to ensure that the voter has completed his/her voting procedure are not explained here.

3) Common Requirements: This layer concerns the analysis, refinement and transformation of high-level requirements to low-level requirements that are split into different parts. The key point of our approach is the definition of finite state machines (FSMs). They describe the voting process and allow to organize requirements in functional areas. Each state machine specifies the possible states of the administration and voting modes in separate state machines. Each state identifies a functional area to which the corresponding use cases and scenarios, and non-functional requirements are provided. The encoding of the FSMs, which are based on UML statecharts, is directly from laws, procedures and principles defined in the previous layer.

The common requirements is one of the innovative parts of this work, because the FSMs used to organize the requirements were also designed in such a way to represent control logic of the actual voting machine. As the development proceed, the FSMs are automatically translated into executable code after they are been formally verified [8]. For this reason, they serve as a binding mean between requirements and development.

Another point to emphasize here is that the relationship between states of the voting FSM and event specific layer. It means (i.e., the relationship) that requirements described in the common requirements layer are general with respect to election specific details. Nevertheless, when needed, requirements in the common layer can explicitly reference which part should be properly customized for the election under consideration. There is a requirement, for instance, which describes the format of electronic ballots. Part of ballot’s data is general (such as, the date, town name, etc) and part that can change according to the type of election (such as, preferences can vary from none, like in referendum, up to two or three options, like in preferential election). This confirms the existence of a connection between the voting state machine and the layout and presentation (e.g. GUI) requirements in the very last layer.

4) Event Specific Requirements: This layer structures requirements to be determined or modified for each type of election. They can be further classified into: hardware configuration, electronic layout (layout and presentation and prototype), and data model (voting rules and data representation).

Hardware specific requirements deal with possible hardware configurations described in the common layer. For example, the voting machine requires an external LCD display whose generic requirements are in the common requirements layer. The actual display, though, can have two or four possible text lines. Therefore, the detail about what
need to be displayed there is specified within the hardware specific requirements.

The meaning of prototype in the picture is to underline the existence of a separate document or mock-up whose layout is approved by electoral officers prior to start the actual implementation. The prototype is actually another representation of the layout and presentation. Therefore it does not require separate requirements, instead it uses the layout and presentation requirements.

Last, data model requirements include descriptions about the counting algorithms or data-interchange formats, which clearly depend by the type of election.

B. Requirements Document

The result of the previously described approach is a requirement document written in natural language and a subset of UML diagrams (such as, statecharts, use cases and scenarios). The structure of this document is organized in chapters, sections, and subsections. For the sake of presentation, however, we proceed as follows:

- Preparatory Part:
  - Introduction
  - Principles
  - General requirements
  - Others (Glossary, bibliography and notation conventions)

- Main Part:
  - Operational logic
  - Functional areas

- Electoral event specific requirements

Figure 2. Abstract view of requirements structuring.
The introduction, principles, and general requirements constitute a preparatory part of the document. This corresponds to the principle layer of Figure 2. The introduction describes the scope of requirements, the voting technology and the voting machine model —i.e., its components from a very high level perspective (e.g., if the voting machine has a printer or not). The principles specify the high-level requirements derived from the procedure (and law) and are valid for any voting or tallying system (e.g., vote anonymity). The general requirements definition, whereas, contains those requirements that span across all the functional areas (e.g., the audit log format).

It is worth remarking that, in some case, general requirements provide a template for the definition of some requirements in the main part of the document —i.e., the operational logic and functional areas. For instance, the subset of general requirements that describe the existence of log information has then to be customized for each use case (e.g., what to log for that use case).

The event specific requirements delve further into details of those requirements that need to be refined depending on the type of election. Hence, each supported election event (e.g., referendum or local election) overwrites a subset of previously stated requirements for which specific information has to be provided.

1) Operational Logic and Functional Areas: The operational logic provides explanation for the FSMs, which are used both in the development of the voting machine software and in the structuring of requirements in functional areas. From the development point of view, the FSMs are used to precisely define the control logic of the voting machine and to automatically generate the corresponding executable code. On the other hand, they provide the rationale of the document structure.

A functional area is derived from a state of the state machines described in the operational logic and is univocally determined by a sequential identifier. The content of a functional area is structured as follows:

- A free text description of the functional area.
- One statechart to further refine the state from which it comes from. (This is optional.)
- A use case diagram to represent the functionalities captured by the functional area.
- A further detail for each use case (see below).

2) Use cases and requirements: Uses cases are characterized by an identifier that depends from its including area and is exemplified using scenarios. Each use case is structured as follows:

- Use case diagram (optional), which decomposes the use case into more elementary pieces.
- Preconditions, which holds when the scenario is applied.
- Scenario, which is a sequence of actions that describe the interaction between users (voters or poll workers) and the voting machine.
- Optional alternative scenarios.
- Exceptional behaviors, which highlights the behavior in case of user mistakes or faults.
- Functional requirements and Non-Functional requirements.

The following template specifies the format for functional and non-functional requirements:

- State, that specifies the current life-cycle state (see Figure 4).
- A free text description.
- Notes, that includes observation, comments and clarifications.
- Example, that clarifies the requirement.
- Type, that can be hardware, software, hardware and software or interface.
- Revision history, a list of items which track the evolution of the requirement. Each entry includes the date when the requirement is changed, an action when the change is made (e.g., added, removed, updated, suspended), a person who did the change and the motivation why a change was needed.

Figure 4 shows the life cycle of functional and non-functional requirements. In particular, when a new require-
The introduction of an ad-hoc tool to manage statecharts and requirements history could overcome some ambiguities. The introduction of an ad-hoc tool to manage statecharts and requirements history could overcome the inconsistency and improve the tractability between the state machines, functional areas and, thus their requirements.

V. MAPPING REQUIREMENTS TO SYSTEM ARCHITECTURE

To provide an idea of the activities we performed and the impact of requirements changes to the architecture, first we summarize the architecture of the ProVotE system in Section V-A (detail can be found [13]). We then detail the mapping between the requirements structure and system architecture in Section V-B.

A. The ProVotE Architecture

For the sake of this work, the ProVotE machine architecture consists of the following components (see Figure 5):

- **The Control Logic.** It defines how the machine has to react to user actions, both in the administration and in the voting mode. The control logic also specifies the logic behind the user interface (e.g., the next screen to show up).

- **The User Interface.** It manages the graphical layout of the administration and of the voting interfaces.

- **The Data Model Management.** It manages all the election specific data, comprising candidates and parties, the ballot data, per-machine election results, and the symmetric and asymmetric keys used for cryptographic operations (such as, ciphering and signing).

- **The Services Component.** It provides the basic functionality to the rest of the application, such as drivers for controlling the external display and printer, logs managers for audits, and managers for transparent, redundant and ciphered persistence of data.

The relationship between administration and voting modes is implemented using the model view controller (MVC) architecture by separating user interface from control logic. The former manages the graphical layout (view). The latter rules the handling of user actions and determines the sequence of screens the voter has to traverse to complete the vote. Moreover, to make the mapping clear, we further distinguish the service components as configuration neutral —to refer to components that are irrespective of the specific election event, and configuration specific —to refer components that can change per election event.

B. Mapping Requirements to System Architecture

In what follows we discuss how the requirements are mapped to different components of the reference architecture shown in Figure 5 (see Table 6).

The architecture of the administration and voting control logic are defined using their respective state machines. This is straightforward from the description given in Section IV. Therefore, the administration FSM is mapped to the administration control logic component of the architecture. Similarly, the voting FSM is mapped to its corresponding control logic.

We can distinguish two kinds of requirements changes on the FSM. The first one is related to the modification of the voting process (which is unlikely according to our experience) or the introduction of a new component. A new component could require the introduction of an additional FSM and can potentially have a major impact on the others. The second case is the addition of new events to an existing state. Suppose, for example, you need to add the possibility to vote a null ballot. This means to modify the state machine FSM and can potentially have a major impact on the others.

Figure 4. Requirement life-cycle.

Figure 5. The ProVotE Voting Machine’s Reference Architecture.
to automatically generate the code for the modified control logic, verify it and write the code to deal with the new event.

The electronic layout (i.e., the layout and presentation) requirements can change according to the election event (e.g., the date of the election, the name of the election district), the ballot design (layout, size, and colors of the ballots to be shown on the screen) and the format of printed ballots (e.g., what to print on the paper ballots, the barcode format). Changes to these requirements are mostly related to the configuration of the Voting User Interface component of the architecture. Therefore, we can map changes to the Layout and Presentation requirements to the Voting User Interface component of the architecture.

As previously seen, the data structures and their internal management is the responsibility of the data model management component, which is separated from the rest of the voting machine logic. The data model requirements describe which ballots are valid for that specific election (e.g., selectable options, the number of allowed choices, if the voter can make his ballot invalid or not) and the internal representation of electronic votes, as well as the counting algorithms that have to be applied. Changes to these requirements affect the configuration properties of the data model component of the architecture. Therefore, the Data Format Specification requirements along with (part of) the Voting Rules requirements are mapped to the Data Model Management component.

The authentication, cryptographic, logging, and persistence are more likely to be neutral services with respect to a change in an event specific requirements. Namely, they should remain unchanged when requirements change per election specific events. On the contrary, the architecture of these components is based on non-functional requirements described in the main section of the requirements document —i.e., the functional areas. For this reason, we can map a change to non-functional requirement to configuration neutral service components. Here we excluded the UPS component because, even if its driver is a service, most of its description is related to hardware requirements.

Finally, hardware specific requirements concern the selection of specific hardware components. The electoral office can choose, for example, from a set of different LCD displays and printers. In that case, the number of rows and columns could vary, which, in turn, affect the amount of information to be shown or printed, respectively, on the screen and printer. Modifying these requirements has a direct effect on the configuration specific components of the architecture. Therefore, the hardware specific requirements as well as (part of) the general hardware requirements are mapped to LCD display and printer components of the architecture.

We remark that the mapping we described is not one-to-one. Namely, one component of the architecture can be defined by set of requirements and the other way round.

VI. RELATED WORK

There has been a great debate on the advantages and problems of various electronic voting schemes. See, respectively, [15], [16], [17], [18] and [19], [20], [21], [22], [23], [24] research efforts to improve the current design of e-voting systems and their critiques.

In [20], [21], [22], [25], [26], assessments of some existing e-voting systems are discussed. These primarily focus on the design and implementation flaws, which could
be exploited to compromise and invalidate elections. The authors also suggest a drastic change in the way in which e-voting systems are designed, developed, and tested.

In [8], [27], [28], [29], the application and usage of formal specification and verification for e-voting systems is given. The authors claim that by applying verification techniques in different levels of abstraction we can build better e-voting machines.

To facilitate and support public administration with tool support, [30] developed a tool to identify, model and analyze existing processes (e.g., paper-based voting process). This allowed them to devise new processes (e.g., e-voting processes) and suggest law modifications.

To our knowledge, so far, organizing and structuring of requirements for e-voting systems is quite an explored area. However, various work have been going on to improve the current specification of requirements in e-voting domain. In [6], the author proposed two approaches to the development of requirements for e-voting systems. Top-down — from abstract to specific requirements, and bottom-up — from scratch to abstract requirements. From our experience, also claimed by the author, this approach can greatly help to balance the over and under specifications problem. In [31] the author presented functional and non-functional (using use cases) requirements for online voting systems. In particular, the authors described the how an online voting system ought to behave. The authors in [32] also proposed requirements and evaluation procedures for a strict subset of e-voting systems by developing new requirements catalog.

VII. CONCLUSION

Managing requirements for e-voting systems is challenging, as it requires to integrate information from different sources, maintaining coherency and integrity. While, at the same time, it allows flexibility in managing software and hardware configurations.

This paper discusses how we tackled the problem for the actual implementation of an e-voting system. One of the characterizing features of our approach is a top-down refinement of the requirements that allows to trace implementable requirements from high level principles and that, in fact, allowed us to implement the control logic of the system with statecharts directly derived from the law and procedures.

The main contribution of this work is two fold: first, structuring of the requirements aiming to bridge the gap between the principles expressed by the laws. Second, mapping requirements to actual architecture of an e-voting system. The separation between generic and election or configuration specific requirements, we proposed in the paper, is concrete and detailed enough to function as a general reference schema that could be adopted by other solutions.

The quality of systems we build depends upon the quality of the requirements that describe them. We believe that converging on a standardized framework for the representation of requirements could help to improve the quality of systems to be delivered. This provides the opportunities for sharing experiences and improving the efficacy of research efforts in the area. We believe that the framework we propose — further refined, standardized, and improved — could be one of the building blocks of such framework.

Possible improvements for the future are the consolidation of the methodology we adopted, tool support (e.g., RequisitePro1), and the abstraction to cover various voting machines and/or voting schemes.

REFERENCES


1http://www-01.ibm.com/software/awdtools/requisite/


