PhD Dissertation

International Doctorate School in Information & Communication Technologies

DIT - University of Trento

Using Formal Methods for Building more Reliable and Secure e-voting Systems

Komminist S. Weldemariam

Advisor:

Adolfo Villafiorita (Ph.D)

Center for Information Technology (FBK-Irst)

February 2010

Acknowledgements

It would not have been possible to write this doctoral dissertation without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Above all, I am heartily thankful to my supervisor, Adolfo Villafiorita. His encouragement, supervision and support from the preliminary to the concluding level enabled me to develop an understanding of the subject. I also would like to say sincere thanks to Prof. Richard A. Kemmerer for hosting me at the University of California Santa Barbara and for his consistent follow-up and enormous feedback.

I offer my regards and blessings to all of those who supported me in any respect during the completion of this dissertation. Particularly I would like to thank Birhanu M. Eshete at the Fondazin Bruno Kessler for the tireless efforts in reading and revising the English of my manuscript, as well as providing valuable comments. I am also grateful to all ICT4G and software engineering groups at the Fondazione Bruno Kessler for their continued moral support. Above all, thank you Chiara Di Francescomarino and Andrea Mattioli for all your kind help since the day I knew you.

Komminist S. Weldemariam

Abstract

Deploying a system in a safe and secure manner requires ensuring the technical and procedural levels of assurance also with respect to social and regulatory frameworks. This is because threats and attacks may not only derive from pitfalls in complex security critical system, but also from ill-designed procedures. However, existing methodologies are not mature enough to embrace procedural implications and the need for multidisciplinary approach on the safe and secure operation of system. This is particularly common in electronic voting (e-voting) systems.

This dissertation focuses along two lines. First, we propose an approach to guarantee a reasonable security to the overall systems by performing formal procedural security analysis. We apply existing techniques and define novel methodologies and approaches for the analysis and verification of procedural rich systems. This includes not only the definition of adequate modeling convention, but also the definition of general techniques for the injection of attacks, and for the transformation of process models into representations that can be given as input to model checkers. With this it is possible to understand and highlight how the switch to the new technological solution changes security, with the ultimate goal of defining the procedures regulating system and system processes that ensure a sufficient level of security for the system as well as for its procedures.

We then investigate the usage of formal methods to study and analyze the strength and weaknesses of currently deployed (e-voting) system in order to build the next generation (e-voting) systems. More specifically, we show how formal verification techniques can be used to model and reason about the security of an existing e-voting system. To do that, we reuse the methodology propose for procedural security analysis. The practical applicability of the approaches is demonstrated in several case studies from

the domain of public administrations in general and in e-voting system in particular. With this it can be possible to build more secure, reliable, and trustworthy e-voting system.

Keywords
procedural security, security assessment, electronic voting, formal methods, specification and verification;

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Statement

As our society is getting more dependent on (computer) system, security is becoming one of the important constraints in system development. However, security is a complex non-functional requirement whose implementation requires intervention over the entire parts of a system at all levels of detail. It is widely reported that all systems and system processes making use of Information and Communication Technology (ICT) are exposed to security threats in one way or another (Avizienis et al. 2004, Myagmar et al. 2005). The security threats are meant to compromise or invalidate the confidentiality, integrity, and availability of the systems and system processes.

A secure and reliable system, if it exists, must satisfy its critical requirements. In order to assess and certify whether a given system meets such requirements, a security analysis or assessment is conducted. To carry out the assessment in a systematic way, usually a methodology is devised by attaching context information to the system under analysis. Five constructs are pivotal to define a methodology for security assessment of a system: asset, threat, vulnerability, attack, and risk (Bishop 2002, Rogers et al. 2004, Common Criteria 2007).

An *asset* is defined as something that has a value to the relevant organizations or business firms. A *vulnerability* is a weakness in the architecture, design or implementation of a system. A *threat* is a potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm. An attack can be identified as the entire process allowing a threat agent to exploit a system by the use of one or more vulnerabilities. A risk may be defined as the probability that a damaging incident is happening (when a threat occurs because of a vulnerability), times the potential damage. The first step in security analysis of a system is, therefore, to define the threat model using these key constructs. The threat model for a system describes the goals an attacker might have, the types of attackers that might attempt to attack the system, and the capabilities available to each type of attacker (Jones and Ashenden 2005, Vraalsen et al. 2005, Vetterling et al. 2006).

Electronic voting (e-voting) systems are increasingly replacing the traditional paper-based voting systems, by making the voting process more convenient and may therefore lead to improved turnout (Gritzalis 2003, Mercuri and Camp 2004, Sastry et al. 2006, Anane et al. 2007, Bishop and Wagner 2007). Electronic recording and counting of votes could be faster, more accurate, and less labor intensive. However, they are fundamentally used in environmental conditions that are quite peculiar, if not unique. In fact, they are probably the only complex safety and security critical systems for which all these conditions are met: they run in an environment with limited control; they are operated by people with the most diverse training and experience; they have to be accurate, while, at the same time, have stringent requirements related to what they can trace, to protect secrecy and anonymity. Logistics and support, due to the geographical distribution and time constraints of elections, add further complexity.

The advantages that e-voting systems can bring cannot be achieved

without an observable cost (e.g., risks) one of which is opening security vulnerabilities for attackers (Mercuri 2001, Prosser et al. 2004, Bishop and Wagner 2007, Rivest and Wack 2006, Gardner et al. 2007). In that respect, recently we have seen that several currently deployed e-voting systems (mostly, the direct recording electronics (DRE) voting systems as used in U.S.) share critical failures in their design and implementation, which render their technical and procedural controls insufficient to guarantee trustworthy voting (Lambrinoudakis et al. 2003, Kohno et al. 2004, Bryans et al. 2006, Gardner et al. 2007, Balzarotti et al. 2008). Moreover, elections are trust situations in which the trust properties reflected both on individual and system level, independently of the voluntary, organizational (or the government itself) or stringent nature of the procedure. Trust in elections could, in principle, be established, if and only if, citizens convince about security with respect to the system and its procedures are respected and followed correctly. The lack of trust can also render even most secure and most reliable systems completely useless (Oostveen and den Besselaar 2004, Antoniou et al. 2007).

The introduction of a new technology and the transition to a new form of systems changes risks and attacks that can compromise or invalidate the security goals. Therefore, the development and verification of fair and secure e-voting systems (in general, any system) should not only take into account their inherent features (e.g., what kind of protection they offer to tampering with), but also how they fit into the environment in which they are used, namely in elections. This leads to a demand for good security risk analysis methods, which must handle detailed analysis of technical aspects as well as more high-level procedure or organizational level analysis.

The research question we investigate in this dissertation is the following:

> *How can we build more secure, reliable, and trustworthy e-voting systems?*

Typically, there are a number of established approaches to tackle the above stated problem by modeling, specifying, and verifying a system satisfies a set of properties (Fredriksen et al. 2002, Avizienis et al. 2004, Xu and Nygard 2005, Myagmar et al. 2005, Wimmel 2005). The integrity and assurance of a complex and safety-critical system's correct behavior with respect to modeling and specification can be achieved, if good engineering practices are appropriately devised and used. With respect to this, the usage of formal methods has been shown to improve the security and quality of complex systems (Kemmerer 1990, Hall 1990, Lowry and Dvorak 1998, Heitmeyer et al. 2008). These approaches allow designers to prove, test, or otherwise examine interesting properties of a complex process whose behavior is specified abstractly, and then interactively refine the behavioral specification to be as close to an implementation as appropriate for a given assurance level. Unfortunately, their usage in voting in general and in the development of e-voting solutions, in particular, is very limited and unsatisfactory. For that matter, we are only aware of the following works (Kremer and Ryan 2005, Campanelli et al. 2008, Delaune et al. 2009, Sturton et al. 2009, Delaune et al. 2009). These works demonstrate the feasibility of applying formal methods for verifying voting machine logics.

Finally, there are also some more reasons why the existing approaches do not perform well in procedural rich scenario. If a methodology exists, it is difficult to formally represent and analyze some peculiarities, namely in elections. Among these peculiarities we mention the following. *Mobility of assets*, assets and sensible data related to an election are handled (and may be altered) by different actors (e.g., technicians, poll officers, electoral officers) with different responsibilities over time and in different locations. *Evolution of assets*, assets related to an election change their value over time. The effects of an attack on an asset change dramatically according

to the period in which the attack is performed (e.g., tampering with an e-voting machine after the election does not have much of an effect). *Number of instances*, various electoral assets need to be replicated for running an election. The effects of an attack may not only propagate to copies, if the master is compromised, but may also have a different impact, depending on the number of instances that are affected by the attack (e.g., breaking one e-voting machine may not have a tangible effect on an election).

1.2 Aims and Approach

In this dissertation we show that it is feasible to improve a system with acceptable security level through the use of interdisciplinary approaches of proper engineering practices. The central concept is to intermingle different techniques for the development of a system. In particular, we have two main aims in this work.

The first one is aimed at providing a technique that allows to understand how the switch to a new technological solution changes security, with the ultimate goal of defining the processes regulating e-voting system that ensure a sufficient level of security for the system as well as for its procedures. Along this line, an important goal is to provide a tool supported methodology within a general-purpose tool, to reduce the necessary effort and experience for the security-related activities and to increase the level of confidence in the development process. Such confidence could have been unthinkable had not we used formal methods. The applicability of the developed approach should also be demonstrated for various case studies of practical relevance. In this research, the approach we propose focuses on the definition of methodologies and techniques to model and formally verify business processes.

The approach we propose includes not only the definition of adequate

modeling convention, but also the definition of general techniques for the injection of attacks, and for the transformation of business processes into representations that can be given as input to model checkers. For this purpose, we describe a methodology related to the security assessment of the procedures. The underlying approach that we follow lies on the intersection of three areas, namely BPR, security, and formal methods. The center of the approach is a security-enriched model of the procedure, where security requirements to be checked are included in the form of logic. We explain our concept using the flows of assets and processes that we recognize as workflow activities, using executable specifications. More specifically, starting from the process model, first we provide formal model of the procedures in terms of transitions system. We elaborate threat-actions using UML diagrams and extend the model by injecting the threat-actions at each possible execution step. The resulting model is then encoded using the NuSMV (Cimatti et al. 2002) input language, and using the NuSMV model checker to help us analyze the effects of attacks on the execution of the procedures. The use of a general-purpose tool makes it possible to communicate and share models among participating parties. As this is an integrated part of the overall system to-be developed and to build on existing tool support and experience on the part of the lawmakers, business analysts, and software engineers.

In contrary, the second aim at investigating on the systematic application of formal methods for reverse synthesis of an existing e-voting system by modeling and verifying its underlying behaviors, then also enriched with attacks, against critical security requirements. The ultimate goal is that of specifying generic requirements for the next generation of e-voting systems. In this research, the approach we follow focuses on the usage of formal methods to study and analyze the strength and weaknesses of currently deployed e-voting machine in order to build the next generation e-voting

systems. We refer to this activity as reverse synthesis. More specifically, we show how formal verification techniques can be used to model and reason about the security of an e-voting system.

Of central importance in our approach is formulating each individual component of the voting system for one of currently deployed e-voting systems named Election Systems & Software (ES&S) e-voting system (Inc. ES&S 2007) as process instance in ASTRAL declarative language (Kolano et al. 1999), along the behavioral specifications of each instances. We specify and analyze critical security requirements about each individual component and the system as whole. We then extend the original formal specification by augmenting a set of transition specifications that represent known attacks, that are needed to model the specific scenarios presented in (McDaniel et al. 2007). Thereafter, we attempt to analyze the extended model against the same set of critical security requirements as the original specification should respect. We use the ASTRAL Software Development Environment (SDE) to process the specification and to automatically generate proof obligations for the PVS analysis tool (Owre et al. 1993, SRI).

1.3 Main Contributions

The main contributions of this dissertation are summarized as methodological contributions for both the modeling and analysis of (system) processes and security assessment techniques, and practical contributions on the usage of the methodologies for the development and assessment of real world e-voting systems.

Methodology for Procedural Security Analysis

Risks and attacks not only depend upon the security levels the new systems offer, but also occur by circumventing the controls and procedures

regulating the way in which the systems are operated. Therefore, introducing technical security mechanisms —such as (Adida 2006, Sastry et al. 2006, Yee 2007, Santin et al. 2008), to secure systems and system processes is not sufficient in the situation where the usage of the system is (totally) influenced by legal framework. For this reason, it is necessary to revisit the way in which the development process are carried on, as well as the goals of security and security threats are specified, modeled, and analyzed. To fill this gap, we introduce what we call procedural security analysis. Since asset mobility, state, evolution, and the context in which asset instances are used in electoral system are an inherent challenge, we introduce a generic assets-centered methodology for procedural security analysis.

Interestingly, the methodology we devised allows for systematic analysis of procedural security based on explicit reasoning on asset flows. That is, by building models to describe the nominal procedures under analysis and injecting possible threat-actions by assuming that any combination of threats can be possible in all steps into such models. We also outline encoding strategies using NuSMV input language —that it is amenable for formal analysis allowing to reason on different properties of the procedure on the extended model.

The benefits that the proposed methodology brings are twofold. First all of, it helps identifying the security boundaries, that is, the conditions under which procedures can be carried out securely. More specifically, using the NuSMV model checking facility, it is possible to understand what are the hypotheses and conditions under which a given security goal is achieved or breached. Secondly, it helps devising a set of requirement to be applied both at the organizational level and on the (software) systems used to make systems and system processes secure. This can be achieved by analyzing the generated counterexamples by the NuSMV analysis tool, since counterexamples provide information to try and modify the existing

procedures so that security breaches are taken care of. Our work takes step forward the motivations for procedural security discussed in (Xenakis and Macintosh 2004a, 2005a). Additionally, we believe that, such results are foundations to familiarize actors (election officials or polling officers) with the possible procedural threats and attacks that can happen during elections, or otherwise anytime during the electoral phases, and thus complement works like (Volkamer and McGaley 2007, Volkamer 2009).

Formal Methods by Reverse Synthesis

We show how formal methods can be effectively applied for the specification and verification of an e-voting machine, using the connection of ASTRAL SDE and the PVS analysis tool.

The two main lessons drawn from this work are:

1. *Formal methods help get a better understanding of the security "boundaries" of e-voting systems.* In the case of the ES&S, for instance, various security requirements could not be proved without making assumptions about the procedures and about the environment. Notice that we expect this result to equally apply to other e-voting systems. Formalizing such hypotheses helps to delineate the necessary conditions for the secure use of systems.

2. *Open specifications play a pivotal role for the development of more secure e-voting systems.* The formal specification of the ES&S, for instance, required the collection of information from different sources, such as configuration instructions, the user's manual, and videos. The adoption of an open-standardized specification could help simplify, extend, and generalize the results we have found to other systems. Based on the current results, we would like to provide a *generic* specification for DRE-based e-voting machines. This generic specification could

then be used as a basis for the specification and design of a new generation of an e-voting system.

Additionally, this work makes the following specific contributions: First, this work advances the current usage scenario of formal techniques in e-voting systems in which procedures and environmental factors also play vital role for a correct execution of the system. Secondly, this work is the first attempt adding on top of a system level testing and analysis of e-voting systems, e.g., (Kohno et al. 2004, Sastry 2007, Ansari et al. 2008, Balzarotti et al. 2008) in which complex low level engineering techniques have been utilized for analyzing and verifying e-voting machines; all these provide only (a relatively) low level of assurance.

Tool Supported Methodology for BPR

We elaborate a generic tool-supported methodology for BPR in public administration for the construction of process models from domain experts or from the laws that describe the procedures. The methodology was originally proposed in (Mattioli 2006). We extended by refining, structuring, and adding some definitions and formalizations for some concepts. Using the methodology, it is straightforward to obtain an unambiguous, standard, general, and objective process models about the procedures and laws under analysis. The methodology also allows to link the laws and models in order to increase the traceability between them. Traceability helps law makers elaborate models in collaboration with software developers or process engineers, and understand the impact of law or process changes as a result of interventions due to people, processes, and some kinds of technologies. When a change is made to the law, being able to identify which processes are defined (or regulated) by the modified part of the law allows us to modify the process model accordingly. By looking at the model, it is

then possible to determine what processes "interact" with the processes affected by the change in the law. The modification can then be propagated to all the relevant processes and makes the model up to date. On the other hand, the reengineering of processes may result in a need to modify some parts of the law. Maintaining law-model traceability allows to automatically identify which parts of the law should be amended by tracing back to the parts of the law that originally defined the modified processes.

Evaluation using Case Studies

The methodologies along with the supportive tool have been applied in two Italian territories that have autonomy over local elections and have experimented a transition to e-voting. Specifically, for the modeling of the electoral law and procedures in the Province of Trento and the laws for the introduction of an e-voting system for a local poll in two municipalities of the Autonomous Region of Friuli Venezia Giulia. Within the project we have analyzed the existing laws and procedures, defined the procedures and developed the (e-voting) system that has been used in local elections —mainly for experimental purpose. The results of the modeling and analysis led to enrich and improve the business process models by re-organizing voting processes that define the *to-be* system description for ProVotE e-voting system, and consequently for the development of the system itself, by helping eliciting, structuring, as well as maintaining requirements for the ProVotE e-voting system —see also in (Villafiorita et al. 2009b,a, Komminist Weldemariam and Mattioli 2009). Furthermore, the ES&S e-voting system is used to demonstrate our reverse synthesis approach.

1.4 Organization

The remaining part of the dissertation is organized as follows:

- Chapter 2 discusses an overview over the current state of research in the following areas: BPR, security and risk analysis, formal methods, and elections and trends in e-voting systems.
- Chapter 3 discusses a generic methodology and supportive tool for BPR in public administration for the construction of (business) process models.
- Chapter 4 presents the procedural security analysis and a methodology for the modeling, encoding, and analysis through case studies.
- Chapter 5 presents the reverse synthesis approach of an e-voting system using ASTRAL language and PVS analysis tool.
- Finally, Chapter 6 draws some lessons learned during our research activities and outlines some possible directions for future work.

Chapter 2

State of the Art

The ideas in this dissertation are built upon existing works in the business process modeling, security, and formal methods fields (see Figure 2.1). We apply known principles and original ideas to a new and socially important problem domain —namely, elections. In this chapter, we discuss current trends in these fields and how they can be adopted for the development of safe and secure e-voting system. Thus, first we review business process modeling and analysis approaches. Second, we discuss security analysis approaches. Finally, we overview elections and e-voting, and the current trends on the usage and application of these fields for the development of e-voting systems. Moreover, we provide an overview of an e-voting system named ProVotE.

2.1 BPR Context

Business process is a set of logically related tasks performed to achieve a defined business outcome. A process is a structured, measured set of activities designed to produce a specified output for a particular customer or market. It implies a strong emphasis on how work is done within an organization. The model describing how the processes interact is called business process model (simply, process model). Thus, Business Process

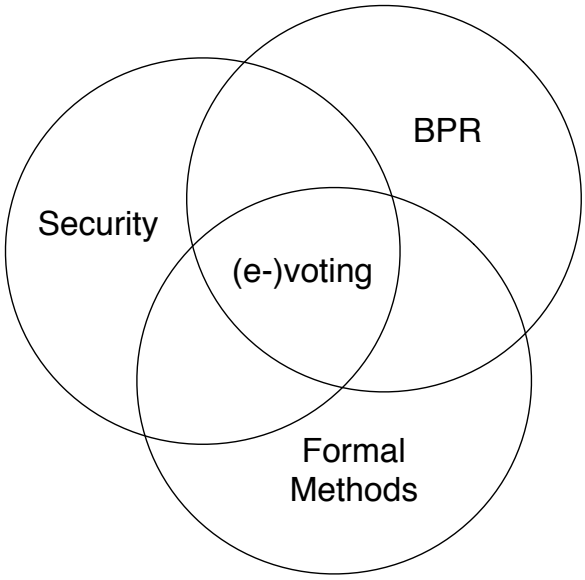


Figure 2.1: Three research areas.

Reengineering (or Redesign) (BPR) is *the analysis and design of workflows and processes within and between organizations* (Davenport 1993, Davenport and Short 1990, Malhotra 1998, William J. Kettinger 1997).

BPR helps an organization to better achieve its strategic goals by acting on three dimensions: people (and their skills), technology, and processes. We distinguish two kinds of process models: business architecture (as-is) and software delivery (to-be). At a very high level and with several simplifications, BPR consists of the following steps, along the line of (Wastell et al. 1994, Valiris and Glykas 1999, Gordijn et al. 2000): modeling the current processes (the as-is), defining goals of the re-engineering activity (e.g., increase efficiency, reduce costs, introduce a new function), devising new processes (the to-be), and actuate them.

Both the as-is and to-be can be represented in some textual or graphical representations. In the implementation, the as-is situation (e.g., actual skills, processes, and systems) is the starting point that might limit the scope of intervention. Notice that based on the requirements of the definite process design, an outline of the technology components must be

made, as well as their interaction patterns. This is the architecture of the new process. Furthermore, systems and applications have to be designed enabling the new process to function according to its detailed design. Both the technology architecture and the applications have to be actually constructed and tested. Finally, when we look at process re-engineering in addition to the constraints, we need to take into account also the law that regulates the goals, missions, and often, also the way in which work has to be organized.

Several strategies have been proposed in the literature to understand, model, and analyze business process models. Three aspects are central in these approaches. The first is tools used for creating (business) process models. Second, notations used to represent the modeling elements and concepts. Third, techniques used for formally specifying and verifying how such models respect the intended goals. For instance, various works in the past have been proposed for modeling business processes. These approaches span from workflow nets to event-based process chains, from flow-charts diagrams to UML Activity diagrams (ADs) and Business Process Execution Languages for Web Service (BPEL4WS)(Juric 2006) and several other works such as (Jintae Lee 1993, Lehman, M. M. 1997, Hlupic 2003). In particular, (Dumas and Hofstede 2001a,b, Castela et al. 2001, Eshuis 2002, Russell et al. 2006, Schattkowsky and Forster 2007) widely discussed the usage of UML ADs for modeling business processes as well as workflow modeling and specifications.

Similarly, a number of approaches focused on specifying and verifying business process models. These include automata, process algebra, and Petri Nets. Each equips with manual and/or automated analysis techniques. Automata based approaches are common, which comprise of a set of states, actions, transitions between states, and an initial state. Labels denote the transition from one state to another. The NuSMV and

SPIN/Promela[1] (Mauw et al. 1998) for instance, are derived from automata to express system behavior in terms of transition systems. Figure 2.2 shows a typical scenario of formal methods in business process modeling.

Petri Net (Peterson 1977) is widely applied for modeling workflow specifications. It allows identifying several basic aspects of concurrent systems/processes simply, mathematically and conceptually. For this reason, it often become a topic in business process modeling for capturing process control flows and a topic in web service process composition for BPEL4WS (Wohed et al. 2003). In BPEL4WS, moreover, Petri Net is specially applicable for detecting the dead path of business process models whose preconditions are not satisfied. The application of process algebra for process modeling is demonstrated in (Salaun et al. 2004, Salomie et al. 2007, Zhan-Haomin et al. 2008). Salaun et al. (2004) particularly discussed the usage of process algebra to describe, compose, and verify interaction of web services when they are composed together. General requirements for the formalization of workflow models using process algebra can be found in (Hofstede and Orlowska 1999). Other works also focused on integrating formal methods —e.g., using π-calculus (Puhlmann and Weske 2005, Ma et al. 2008) and SPIN (Holzmann 2003), with business process modeling aimed at specifying, simulating, and verifying one's business process models are designed. The results of the verification can be analyzed to improve the models, thereby improving the actual business process. These techniques can detect and correct errors of the models as early as possible and in any case before implementation.

A formal model for business process modeling and design is discussed in (Koubarakis and Plexousakis 1999, 2000), with emphasis on representing knowledge about organizations and their business processes formally, inspired by the work presented in (Loucopoulos and Kavakli 1995). Their

[1] `http://spinroot.com/spin/whatispin.html`

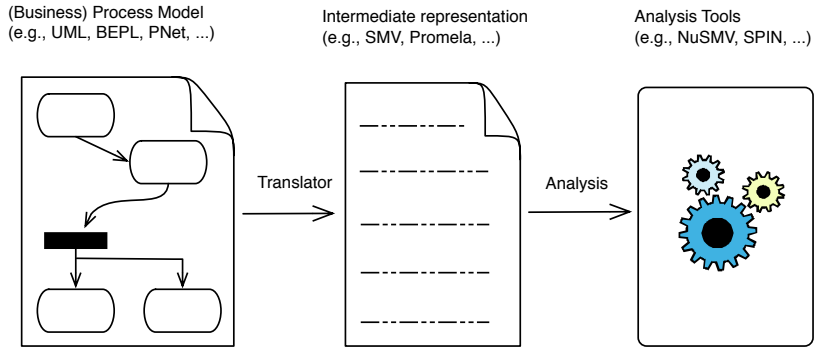


Figure 2.2: Typical Scenario of formal verification in business process models.

formalization is based on the use of situation calculus (Levesque et al. 1998) and the concurrent logic programming language ConGolog (Giacomo et al. 2000) for representing knowledge about organizations and their processes. More specifically, their approach allows to develop the so-called enterprise model. It comprises five interconnected sub-models to formally describe different aspects of an organization. The core elements that constitute the enterprise model include actors with their roles, goals, process (distinguished between primitive and complex actions), enterprise entities, and constraints. A goal-oriented methodology for business process design is also outlined for developing a new business process. The ConGolog formal specification is developed as a set of sub-models to capture the new process from various viewpoints. Formal verification can be carried-out to check, for instance, whether each role responsibility is fulfilled and each constraint is maintained by the ConGolog procedures defined for each individual role.

The usage of model checking for verifying functional requirements on workflow specifications is discussed in (Eshuis 2002, 2006). To specify workflows, the author used UML ADs by defining a formal semantic for them in order to meet the semantics of the target specification language that is suitable for formal analysis. Two kinds of semantics are specifically introduced for the ADs, namely a requirements-level and an implementation-level semantic (Ch. 2 and 3 Eshuis 2002). A number of translation rules have been

defined to convert AD nodes (such as activities, objects, data and control flows) into NuSMV input language semantics, such that functional requirements about the business process models can be model checked. To support their approach, the authors present a tool in (Eshuis and Wieringa 2004), so that the models specified in the ADs are translated into the NuSMV input language for model checking.

An approach for the specification and verification of artifact behaviors in business process models is presented in (Bhattacharya et al. 2007, Gerede and Su 2007, Gerede et al. 2007). The center of the approach is the artifact-centric operational modeling for constructing models with an appropriate formalism. The three key constructs of the approach are business artifacts, business work descriptions —describe the workflow activities on an artifact by which a business role adds measurable business value to this artifact, and repositories. These constructs define the operational modes of the modeling, where each of the construct is represented formally. Differently from the verification goal discussed in (Koubarakis and Plexousakis 2000), which focuses on the use of formal methods for verifying properties about only processes in business process of an enterprise, with this approach the verification goal is checking whether models satisfy certain artifact properties. Examples of properties they claim to verify are reachability and arrival of artifacts into a repository in bounded domains. The authors also presented a logic language based on CTL named Artifact Behavior Specification Language (ABSL). ABSL allows to specify the lifecycle properties of artifacts, where the lifecycle of an artifact type specifies the possible sequencing of services that can be applied to an artifact of this type as it undergoes through the business process. Yet, the same concept like above cited with some extensions enabling to automatically construct business processes is presented in (Deutsch et al. 2009, Fritz et al. 2009).

However, the authors (of the artifact-centered approach) did not show

how to perform automated analysis or verification nor hinted the integration of their approach with existing formal methods' tools. Mostly their focus is in constructing business process models with correct creation and termination of artifacts during their lifecycle, as opposed to the paradigm of organizing and modeling workflow or business processes around relatively flat process-centric models. Additionally, they hardly speak about security analysis.

We have discussed various works demonstrating their usage scenarios insofar as they can used for modeling, specifying, and analyzing business processes and workflows. Unfortunately, the attempt to model laws and procedures, as well as to perform formal analysis in favor of the public administration (PA) is not satisfactory. The advantages and difficulties related to the (re-)engineering of PA can be found in (Wastell et al. 1994, Alpar and Olbrich 2005).

In recent years, however, a number of governments have been adopted such techniques to support their PAs. Works like (Willcocks et al. 1997, Thaens et al. 1997, Greco et al. 2005, Lenz and Reichert 2005) particularly have been discussed BPR support for public healthcare services by identifying different levels of process support and by distinguishing among generic process patterns. The use of BPR for better government has also been discussed earlier by the U.S. federal government and the U.S. Department of Defense and its use in taxation in (Review 1993). The importance of modeling in the legal framework and documenting the knowledge about the legal constraints within the process model itself is stated in (Alpar and Olbrich 2005).

In particular, in (Olbrich and Simon 2008), the authors discussed an approach based on event-driven process chains and suggested how to translate law paragraphs into process models using the Semantic Process Language (SPL). Their main goal is to the visualization and formal modeling of a

legally regulated process. The interesting aspect of this work is not only the consideration of the given law when developing business process models, but also the explicit derivation of a process structure which is implicitly specified within the paragraphs themselves using the SPL. The SPL enabled them to articulate language structures into executable workflow models, using Petri Nets. The presented approach could provide means for verifying whether process-like behavior fulfills the selected paragraphs formally.

2.2 Security Analysis

As defined by ISO/IEC 15408 (Common Criteria 2007), threats to security (or security threats) are potential attacks that misuse or breach the security goals. Security threats exist when there is an opportunity, capability, action, or event that could violate security. Security threat modeling (simply threat modeling) and analysis are (semi)formal mechanisms for identifying, documenting, and assessing the risks caused by security threats. In the literature, there are several approaches and techniques to perform security analysis. These include, but are not limited to graph-based, model-based, formal methods, and a combination of one or more of these techniques. What is common to all, however, is that they include five key constructs (assets, threats, attacks, vulnerabilities, and risks), during their modeling and analysis approaches. In the following, we discuss some of existing works in security analysis relevant to this work.

2.2.1 Graph-Based Approach

In the graph-based approach, the description of an attack is provided in the form of graphs that describe the logical sequences and requirements with which a malicious actor composes attack actions. The most widely

used representation for attack modeling and analysis using graph-based approach is attack tree, a term coined by Schneier (Schneier 1999, 2004). Attack tree is a well-structured methodology for analyzing the security of systems, subsystems, and system processes —and, it is used to capture the sequence of attack actions and their interdependencies among them. It provides a way to think about security, to capture and reuse knowledge about security, and to respond to changes in security.

Quoting from (Schneier 1999):

> *"Security is not a product, it is a process, and attack trees form the basis of understanding that process."*

In the attack tree, all the top-level nodes represent the ultimate goal of the attack and each lower-level nodes represents the actual executable attack action. Attack trees focus on generic description and sequences of attacks following the AND/OR logical rules. However, a larger number of attacks can carried out both through the exploitation of the knowledge of the system and multi-action coordination (Steffan and Schumacher 2002). (Braynov and Jadiwala 2003) proposed a formal model of attack trees in terms of actions in order to understand, represent, and analyze such coordinations. The model accounts for synergy and coordination among attackers, by allowing specifying concerns like the actor in charge of performing the action, the list of action preconditions, other concurrent actions, and one or more action postconditions.

An approach to automatically generate and analyze attack graphs using formal techniques is discussed in (Sheyner et al. 2002, Sheyner 2004). Their aim was that of examining, and eventually discovering design errors using model checking technique (specifically, by using the SMV model checking facilities) for the verification of a finite system model against a formal correctness property specifying acceptable behaviors. Fovino and

Masera (2006) introduced a multidimensional view of attack trees allowing to attach context information to the boundary knowledge (knowledge about the system, such as the weaknesses of the target system, security properties of the system, and potential threats) on each attack tree. Specifically, the approach distinguishes each node of the attack tree among three main categories: operations (which represent any step operation made by the attacker to perform the attack), vulnerabilities (which describe a vulnerability required to realize the attack), and assertions (which represent assumptions, results, or requirements that characterize the attack process). The approach could allow to use attack patterns during the assessment process while context information is appropriate. Though yet to investigate in practice, this approach could work best with the methodology presented in (Daley et al. 2002).

The use of goal-orientation has been well acknowledged in (early) requirement engineering for eliciting, specifying, analyzing, and documenting (software) system requirements (Dardenne et al. 1993, van Lamsweerde 2001). The approach is also widely adopted for security domain (van Lamsweerde 2004, Elahi and Yu 2007, Yu et al. 2008). The overall idea of goal-oriented security specific modeling and analysis is to use the notion of "goal" to represent the sequence of actions of an adversary where the sequence ordering can be specified appropriately. In this way, it is possible to reason on the possibilities of goals satisfiability or deniability. High level goal (i.e., from attacker's viewpoint) is said to be satisfied when all the subgoals have been fulfilled.

2.2.2 Model-Based Approach

In the model-based approach, a combination of a system or process modeling language and traditional security or risk analysis methods are intermingled. More specifically, model-based is aimed at supporting a tight integra-

tion between security analysis and development process, by constructing models for the system and security threats. The approach, on one hand, helps detecting or identifying early security problems. On the other hand, it provides a systematic way to wave security into the development process. Notice that we are not saying that all model-based approaches are applicable for development process. To make our discussion more clear, thus, we distinguish model-based approaches into two categories: one focuses on security or risk analysis and/or documentation, whereas the other gives more emphasis on security-centric development process.

With respect to the first category, there are a number of specific as well as general approaches. The Common Criteria, CORAS, Risk Analysis and Management Methodology (CRAMM), and Information Security (INFOSCE) Assessment Methodology (IAM) are among the most widely used ones. With respect to the second category, we mention security specific development process, and extensions and applications of UML for security.

The Common Criteria, ISO/IEC 15408

Since 1990, work has been going on to align and develop existing national and international schemes in one, commonly accepted framework for assessing IT security. The Common Criteria (CC) (Common Criteria 2007) represents the outcome of this work. It is an international standard for assessing and validating the security of information systems. The purpose of CC is to allow i) users to specify their security requirements, ii) developer to specify the security attributes of their products, and iii) evaluators to determine if products actually meet their intended behaviors. In addition, it presents requirements for the IT systems of a product or system under specific categories of functional requirements (related to the product) and of assurance requirements (related to the development process). The CC framework establishes a set of seven (where seven in the highest level) Eval-

uation Assurance Levels (EAL) through the association of various features with appropriate assessment methodologies, and provides a structure set for the generation of protection profile (PP)[2] for the target of evaluation.

In CC, the (security aware) development begins by providing a brief description about the target of evaluation and its boundaries. This can be specified either by developers or security analysts. The assets of the target system, the potential threats, the organization security policy, and assumptions and constraints over the operating environment of the target of evaluation are then provided. All these are included in a well-structured form within security target document. The process then describes the countermeasures (in the form of security objectives), and demonstrates that these countermeasures are sufficient to defeat specified threats. The assurance requirements can be used as a means to build confidentiality on the system, because they contain measures to be undertaken during development in order to keep the risk of security threats unlikely. They are necessary for guaranteeing the confidence that the target of evaluation meets its security objectives.

Unfortunately, CC does not say much about methodologies and procedures for specific uses. Moreover, CC leaves too much room for ambiguity and it is difficult to find the right meaning of the security requirements (Mellado et al. 2007). To close this gap, some approaches have been proposed in the past aimed at supporting the CC both methodologically and procedurally. For instance, a process-based framework that describes how to wave CC requirements into the development process at different level of details is discussed in (Vetterling et al. 2002). The authors in (Mellado et al. 2007) proposed CC based security engineering process with main focus on the early aspects of development, to understand and later to derive

[2]A document typically created by users or community that identifies security functional requirements and evaluation assurance requirements plus the trust model for a class of security critical products.

security requirements.

The use of formal methods, with the goal of achieving a high level of assurance earlier mentioned by (Dominique Bolignano 1999). Namely, formal methods are to be used at every stage of the development process (functional, internal, and interface specifications, plus high-level and low-level design). At the EAL5 level, which is the first level for which formal methods can be used, a formal model of the security policy has to be provided so as to verify the consistency of the policy. At EAL7 a formal description of security functions is required. Thus, the use of formal methods should be considered in many cases as the best way to meet the semiformal requirement that are found at lower levels (Bialas 2006, 2007). In line with this, some attempts have been made in the usage and application of formal methods for CC based development aimed at narrowing some of the mentioned gaps see, e.g. (Morimoto et al. 2006, Horie et al. 2006) (Junkil Park 2007, Morimoto et al. 2007, Heitmeyer 2009).

The CORAS framework

The CORAS methodology —first introduced in (Fredriksen et al. 2002), and extended and applied to various applications in (Vraalsen et al. 2005, Vetterling et al. 2006, Hogganvik 2007)— is a combination of UML modeling and structured brainstorming techniques such as HazOp (Hazards and Operability Analysis), FTA (Fault Tree Analysis), and FMEA (Failure Mode Effects Analysis)[3]. The methodology presents descriptions of the target system, its context, and all relevant security features in an easily accessible format. It has three different structures: i) The CORAS risk modeling language, which includes both textual and graphical syntax as well as the semantics of the methodological elements such as the concept of actor, asset, threat agent, threat scenario, and risk; ii) The CORAS

[3]The bibliography references for FTA, HazOP, and FMEA can be found in (Fredriksen et al. 2002).

method, which specifies a step-by-step description of the security analysis process; and, iii) Tool support, a visual tool (named CORAS) for documenting, maintaining, and reporting the assessment results.

The distinct angle of the CORAS approach is that of its emphasis on security and risk assessment tightly integrated in the UML and RM-ODP[4]. In particular, the issue of maintenance and reuse of assessment results is novel and makes the approach attractive. As said before, the CC is generic and does not provide methodology for security assessment. By contrast, the CORAS is devoted to methodology for security assessment. Both the CC and CORAS place emphasis on (semi-)formal specification. However, contrary to the CC, CORAS addresses and develops concrete specification technology addressing security assessment. The CC provides a common set of requirements for the security functions of IT systems, as well as a common set of requirements for assurance measures applied to the IT functions of IT products and systems during a security evaluation. CORAS, instead provides specific methodology for one particular kind of assurance measure, that is, security risk assessment.

Furthermore, the important features from the CRAMM [5] approach are incorporated in CORAS methodology, e.g., assets identification and valuation. Contrary to CRAMM, however, CORAS provides analysis process in which modeling is tightly integrated with the process, not only to document the target system, but also to describe its context and possible threats. Moreover, CORAS employs modeling to document the assessment results from risk analysis and the assumptions on which these results depend. Although the CORAS approach provides the mentioned benefits (e.g., a risk document in the form of risk profile at the end of the methodology), it does not care much about the usage and automation of systematic

[4]Reference Model for Open Distributed Processing `http://www.rm-odp.net/`

[5]`http://www.cramm.com/`

threat actions and their coordination like (Braynov and Jadiwala 2003).

The NSA INFOSEC Assessment Methodology (IAM)

The National Security Agency (NSA) Information Security (INFOSCE) Assessment Methodology (IAM) is a detailed and "systematic" method for examining security vulnerabilities from an organization viewpoint as opposed to only a technical viewpoint (Rogers et al. 2004). It is not a risk assessment methodology; instead it focuses on vulnerabilities and impact of the information system. The security assessment process using IAM comprises of three mandatory phases, and each individual phase has specific objective and output. These phases are pre-assessment, on-site activities, and post-assessment. Within the pre-assessment, there are various activities related to refining organization objectives, understanding criticality of organization information, identifying the system to be assessed and its boundary, etc. By the end of the first phase, an assessment document is produced. In the second phase of the IAM assessment (i.e., the on-site activities), the classical interviewing process from security viewpoint with customers is conducted. The third phase is devoted to the pre-assessment activity, which is nothing but writing the final document of the assessment. The methodology is accompanied by templates, which are designed by the experiences of the NSA security team.

Needless to say that, the IAM does not specify any technique for identifying the vulnerabilities even for those specific vulnerabilities that are from the perspective of organizational concern. Differently from CORAS methodology, the IAM needs experts to perform the assessment including for the on-site activities. In CORAS, this can be done using easy-to-understand visual modeling language by both domain and non-domain experts, during the brainstorming session.

Security Centric Development process

In (Wimmel 2005, Jan Jürjens 2006), security specific development process (security engineering) is presented. That is, the approach attempted to intermingle techniques to ensure secure development of a system. In (Wimmel 2005), in particular, the author defined a model-based development process for security engineering, with a central idea of transforming security-enriched model to a threat scenario by adding an intruder with appropriate capabilities and applying formal methods for security verification and testing on the resulting threat scenario. For creating the security-enriched model about a system of interest, the author extended and used the AutoFocus/Quest tool[6] —a a general-purpose tool and specification language designed for the development of distributed and embedded systems. The formal verification of security requirements is performed on the generated threat scenario using NuSMV model checker, after the threat scenario being transformed automatically into the NuSMV input language.

UMLsec (Jürjens 2002) is an extension of UML notations for a secure system development, by providing means to specifying security requirements. The underlying basis is an abstract state machine model that formalizes a subset of the UML elements and extends with UMLsec specific stereotypes. The purpose is to be able to formally verify software specifications, reduce the number of security risks, thereby making it available to developers who may not be specialized in security. SecureUML (Lodderstedt et al. 2002, Basin et al. 2003) is another extension of UML, that defines a vocabulary for annotating UML-based models with information relevant to access control. It is based on extended reference model for Role-Based Access Control (RBAC) (Osborn 1997), with additional support for specifying authorization constraints. It combines system modeling and system

[6] `http://autofocus.in.tum.de/index-e.html`

security in a detailed level with more emphasis on RBAC. The approach is based on first specifying system models and its security requirements and then use tools to generate the system architecture from such specifications, thereby supporting their integration into a model-based software development process. RBAC is also targeted in (Ray et al. 2004) where the authors model the concept as reusable UML templates. More specifically, by proposing a class diagram template for RBAC and use object diagram templates to specify RBAC constraints.

2.3 Elections and Electronic Voting

It is important to review elections and electronic elections in order to provide some perspectives on the issues and processes involved. However, we will not review the election principles (such as universal, equal, free, secret, direct, trusted) and the majority of technologies used for elections but the DRE technology. These concerns are thoroughly reviewed, investigated, and presented in (Mercuri 2001, Mitrou et al. 2003, Sastry 2007, McGaley 2008, Volkamer 2009).

2.3.1 Elections and Technology

Voters in democracies around the world currently cast their vote in one of a variety of ways. Hence, elections differ quite a bit from nation to nation, not only with respect to the technology chosen to determine the elected candidates (e.g., proportional, majoritarian), but also for the procedures, the way in which votes can be cast, the organizations involved, etc. Looking at Europe, for instance, voting ranges from cases such as that of Estonia, in which voters can cast their vote by mail, through the internet, or by going to the polling station, to that of countries, such as Italy, in which the voting procedures are completely manual and voters are assigned a

specific polling station to cast their vote. The technologies being in use currently span from plain paper ballots to punch cards and from optical scan to paperless technologies. Each of the technology has its own pros and cons[7].

Electronic voting systems are complex distributed systems, whose components range from general purpose PCs to optical scanners and touch screen devices, each running some combination of COTS components, proprietary firmware, or full-fledged operating systems. The DRE voting machines are examples of e-voting systems. They are physically hardened, preventing access to the typical PC connectors, e.g., USB ports. The design of these machines is significantly different, but they can be grouped into three basic types —mostly on the basis of their interface evolution (Coleman et al. 2000).

The first design essentially mimics the interface of a lever machine. In a sense that, the entire displayed ballot is visible at once on the screen. As opposed to the lever machine in which a voter moves levers to make choices, with this kind of design the voter navigates from one screen to the next by pushing a button available on the screen, which in turn triggers an underlying electronic microswitch and turning on a small light next to the choice. In the second design, a ballot page is displayed on a screen, and the voter uses mechanical devices such as arrow keys and buttons to make choices on a page and to navigate among pages. The third type is similar to the second except that it has a touchscreen display, where the voter makes a choice by touching the name of the candidate on the DRE screen and casts the ballot by pressing a separate button.

What is common to all is that, after the ballot is cast the votes are directly stored internally. Moreover, most of the third type of DREs print a

[7]Comparison of Draft Voting Technology Standards Documents: `http://votingintegrity.org/tool/standards/default.html`

paper ballot with the voter's choices listed. The voter could then verify that the ballot accurately reflected her/his choices as made on the DRE screen. This is usually referred as Voter Verified Paper Audit Trail (VVPAT) (or Real Time Audit Log (RTAL)). Thus, the type of DRE equipped with printed audit trails is often called DRE-VVPAT/RTAL. VVPAT was originally suggested by Mercuri (2001). In U.S., these machines are built by private companies, according to various state-specific standards.

DREs are particularly interesting because they simplify a number of complex operational problems (Gritzalis 2003). Many of the reasons for the increased adoption of DRE machines include accessibility and prevention of voter mistakes —e.g., preventing of overvotes and feedback on undervotes before the voter confirms to the final vote; audio interfaces to let the visually impaired vote without assistance; the ballot management is vastly simplified using some kinds of memory cards instead of paper; moreover, DRE machines can save the costs associated with producing and securing paper ballots (Evans and Paul 2004).

As the DRE technology (e-voting technology, in general) evolves, however, many open questions emerge. One of these questions is how to ensure the security of the entire voting chain. Quoting from (Mitrou et al. 2003)

> *"Security primarily refers to the (technically guaranteed) respect for secrecy and freedom, but in reality it covers the entire range of functions and election components such as registration, eligibility and authentication. The ballot being transmitted to the vote counting equipment must be an accurate and non-modifiable copy of the voter's real choice, with no possibility of modification anywhere in the transmission path, in any of the intervening networks and devices."*

This requires to understand a three-dimensional gap: the *technological*

gap —that is, between hardware and software, the *sociotechnical gap* — that is, between social and computer policies, and the *social gap* —that is, between social policies and human behavior, as noted prior by Mercuri (2001). This underlines the fact that e-voting systems are not just to ensure low-level security (i.e., technical guaranteed), but also must ensure all the non-technical (procedurally and/or environmentally guaranteed) aspects of the system and the processes themselves. A lot of efforts are on going aimed at improving the current trends in the development of e-voting machines.

2.3.2 Trends in the Development of e-voting system

Scientific literature on e-voting is wide and multi-disciplinary. We organize previous work in five areas: i) understanding the risks posed by the introduction of e-voting systems in the polling stations; ii) developing requirements for e-voting; iii) assessing existing systems; iv) designing novel voting schemes, protocols and/or techniques; v) applying formal methods for better design of voting machines.

Understanding Risks

With respect to this area, previous work focused on the representation and effective implementation of e-voting procedures, e.g., as business processes. Namely, using BPR for understanding what changes could be introduced in the conventional voting procedures to allow a secure transition to electronic elections. In this area, the premier work is presented in (Xenakis and Macintosh 2005b, 2007) where the authors investigated the need for applying business process re-engineering to electoral process in order to proposed a possible transition into electronic voting system.

Xenakis and Macintosh (2005b) express the need for BPR for the effective administration of e-voting specifically in the following statement:

> *"One can identify the need to redesign the electoral process into a full-scale e-electoral process, delivered through simultaneous multiple technological channels, all contributing to the formation of a unique election result. As such, the electoral process can be considered as a "business process". Subsequently, many of the business management functions can be applied to it in order to manage, control and re-engineer it."*

The PBR concept pertains to the redesign in the context of existing business rules, such that the introduction of e-voting solution can be evaluated. As it is critical to define roles and responsibilities within the e-voting process which could furnish a better understanding of who is responsible for doing what in the different process stages to produce election result, it is also equally important to provide systematic methodology to reason on what can go wrong on this procedural rich workflow, instead of detecting the weaknesses until well after attacks have been taken action(s). In this regard, (Xenakis and Macintosh 2004b,c) discussed these kinds of risks and difficulties while introducing e-voting solution. Furthermore, the same authors in (Xenakis and Macintosh 2004a, 2005a) discussed the need for procedural security in electronic elections and provided various examples of procedural risks occurred during trials in UK. The approach can obviously highlight some of the security implications of the administrative workflow in e-voting, such as, discussed in (Lambrinoudakis et al. 2003).

The authors in (Bryl et al. 2009) presented an approach to reason on security properties of the *to-be* models (which are derived from *as-is* model) in order to evaluate procedural alternatives in e-voting systems. In particular, using Datalog (Eiter et al. 1997) and the underlying theorem prover they expressed and verified security concerns (such as delegation of responsibility among untrusted parties, trust conflict, and so on). The aim is that of understanding problematic trust/delegation relationships and eventually

finding ways to adopt a solution to the detected security properties violations.

As a matter of fact, none of the existing works in this area provide the actual low-level detail such that the realization and analysis for the transition to e-voting is evaluated. The proposed approaches are poor in using techniques to formally analyze what security breaches may be derived by executing the procedures in the wrong way.

Developing requirements for e-voting

There are a variety of international documents such as European Union (EU) Venice Commission recommendations (Council of Europe 2004) and the U.S. Federal Election Commission (FEC) Voting Systems Standard (VSS) (Federal Election Commission 2002, 2005), that describe a set of principles for voting systems. These documents mainly specify high-level abstract principles about the behaviors of each component of a voting system should respect, as well as the related procedures. The FEC-VSS, for instance, 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. However, these kinds of requirements often make difficult the development and implementation of the actual e-voting system. Moreover, the way these documents describe (security) requirements is hard to understand, and sometimes they contain contradicting/conflicting requirements; specifically, the conflict between the requirements for secrecy and accuracy. If the e-voting system needs to be developed in a safe and secure way, there must be an appropriate requirements definition for it.

We acknowledge the following premier contributions (Mercuri 2001, McGaley 2008, Volkamer 2009, Villafiorita et al. 2009a) with respect to requirements development for e-voting system. Besides the technological

gaps that can cause threats to e-voting system, Mercuri (2001) earlier indicated that the (e-)voting implementations present further difficult challenges for realization of the system due to the legal and sociocultural gaps.

McGaley (2008) expresses the lack of adequate requirements in e-voting in the following statement:

> *"The lack of an adequate requirements definition for e-voting prevents us from determining the quality of a given system, and is therefore a barrier to the use of e-voting for critical elections."*

Thus, an essential activity to ensure e-voting system (any system, in general) behaves correctly is laying down what behaving correctly means for that system. This cannot be achieved without a proper engineering approach such as requirements engineering techniques. McGaley attempted to address the mentioned problems by proposing a methodological approach for analyzing the root causes of the conflicts, organizational barriers (or procedural barriers), and requirements for critical election. Her approach focuses on the development of requirements for e-voting with top-down and bottom-up strategies. The first one is aimed at developing a set of requirements from an existing catalogue. The latter, instead is aimed at developing of a new catalogue.

Volkamer (2009) takes forward the development of requirement catalogues by providing standardized, consistent, and exhaustive list of requirements for e-voting systems —specifically, for stand-alone DRE and remote e-voting systems. These requirements not only describe requirements that the system should meet, but also specify the corresponding laws or regulations for the evaluation of the systems themselves. The author developed a methodology for the requirement development process. The results of the methodology include system requirements (divided into functional, security, usability requirements), organizational requirements,

assurance requirements for both stand-alone DRE voting machines and remote e-voting systems. Furthermore, the methodology comprises of cross checks existing catalogues, election principles, and the possible threats. This could allow software engineers and developers to easily understand how their system meet these requirements. Following that, the author proposed evaluation and certification procedure mostly for remote voting system by complementing the CC common evaluation methodology and also developed protection profile[8] for remote voting (Ch. 6 and 7 Volkamer 2009).

Designing novel voting schemes and/or protocols

Prior works with respect to this area focused on the design of cryptographic schemes, protocols, and/or techniques for better designing of voting machines. Cryptographic techniques offer the promise of verifiable voting without needing to trust the integrity of any software in the system (Fujioka et al. 1993, Karlof et al. 2005). In other words, they are meant to augment e-voting machines and provide voters with an end-to-end guarantee of the proper tabulation of their vote. In general, the ultimate goals of these kinds of approaches include: ensuring a voter can be certain that his vote has been recorded accurately (voter verifiability), no voter can prove to anyone else how s/he cast (receipt freeness), an independent body can verify that the recorded votes exactly match with the published tally after the election (Iversen 1991, Cramer et al. 1995, Chaum 2004).

In this area, we mention the following three cryptographic schemes: PunchScan (Carback et al. 2007, Essex et al. 2007), Prêt á Voter (Xia et al. 2008, Ryan et al. 2009), and Scratch & Vote (Adida 2006) schemes. What is most common to all these approaches is that they rely on the

[8] The scope and definition of the PP for online voting systems can be downloaded from`https://www.bsi.bund.de/cae/servlet/contentblob/480286/publicationFile/`

underlying cryptographic principles to various degree of complexity.

PunchScan is a voting system which is easy to use by the voter as well as by election officials while, at the same time demonstrating transparency and reliability. It also provides public verifiability, election integrity, and enhanced voter privacy. Scantegrity (Chaum et al. 2008, 2009) is a successor of PunchScan that meets industrial standard by providing end-to-end verifiability of the integrity of critical steps in the voting process and election results. Prêt á Voter (verifiable electronic elections) is a type of electronic voting system that uses paper based ballot forms that are converted to encrypted receipts to provide security and auditability, at the same time by keeping coercion resistant and making easy to use.

The Scratch & Vote is another cryptographic voting method proposed by Adida (2006). It provides public election auditability using simple, immediately deployable technology. The method combines a variety of existing cryptographic voting ideas such as homomorphic encryption — e.g., which allows votes to be tallied without decrypting individual votes, the cut-and-choose at the precinct approach, and so on. Additionally, works like (Fujioka et al. 1993, Benaloh and Tuinstra 1994, Ray et al. 2001, Santin et al. 2008) attempt to provide (maximum) secrecy and/or anonymity for the vote and voter.

Unfortunately, we cannot say that cryptographic schemes and/or protocols address the current situation in the democratic process for several reasons. For example, the protocols that have been proposed so far do not yet overcome all of the barriers to their use in critical elections (McGaley 2008, page 44); although DRE machines are most popular in public elections in some U.S. states, the applicability and scope of the proposed schemes are very limited in these machines. Moreover, as noted in (Karlof et al. 2005), some cryptographic protocols live with some security holes, such that sensitive information about the election can be leaked in one or

another way.

Sastry (2007) presented the concept of *"designing voting machines for verification"*. The aim of this work is that of providing techniques to help vendors, independent testing agencies, and others verify critical security properties DRE voting machines. The basis idea of the approach consists of two interesting techniques. The first focuses on creating a trustworthy vote confirmation process, where the proposed architecture splits the vote confirmation code into a separate module whose integrity is protected using hardware isolation techniques. The second focuses on helping ensuring a very important property in voting, that is, "None of a voter's interactions with the voting machine, including the final ballot, can affect any subsequent voter's sessions." (see Sastry 2007, page 31). In order to do that, the author proposed hardware resets technique that restores the state of modules components to a consistent initial value between consecutive voters. With this, it could be possible to eliminate the risk of privacy breaches and ensure that all voters are treated equally by the systems.

Other works, such as (Sastry et al. 2006, Yee 2007) apply techniques used in other domains —like pre-rendering user interface and hardware separation— to build a higher assurance e-voting systems with accessible, verifiable, and secure voting system. The design of a trustworthy voting system by exploring the TPM (Trusted Platform Module) infrastructures (e.g., PKI, hardware protection of cryptographic keys) is presented by Paul and Tanenbaum (2009b,a). They presented a scheme that improves registration integrity, and introduced a design that prioritizes election integrity. Their voting system has a nine-step as a whole, which takes place from an election's inception to its final conclusion.

Assessing exiting e-voting systems

With respect to this area, assessing existing systems, some e-voting sys-

tems currently deployed in elections have recently undergone a thorough and independent scrutiny to evaluate their security and quality. This is because, in the recent years, the DRE machines raised serious security concerns. They make the election process less verifiable and greatly expand the aspects of an election for which voters must rely solely on trust. Security vulnerabilities have been reported in each dimension of security gap. These vulnerabilities have been practically investigated and proved by various academic researches. This creates an enigma in the trustworthiness of the machine and the voting process as well.

In line with this, we mention several academic researches (Jones 2003, Kohno et al. 2004, Gardner et al. 2007, Balzarotti et al. 2008, Ansari et al. 2008), that assess both hardware and software of different forms of e-voting machines (e.g., Diebold/Premier, ES&S, InterCivic), used in some U.S. states. In these studies, the researchers identified serious design and implementation flaws that are notable for their level of egregiousness. More specifically, their analysis have showed that the current e-voting systems are vulnerable to very serious attacks, as well as they have produced a catalogue of vulnerabilities and possible attacks. Some analyzes also suggested a drastic change in the way in which e-voting systems are designed, developed, and tested (e.g., by identifying procedures to eliminate or mitigate the discovered issues, by developing a precise methodology and toolsets for the assessment). The assessment approach presented in (McDaniel et al. 2007, Balzarotti et al. 2008) is particularly astonishing; in our opinion, it can be used for other complex-security critical systems evaluation and assessment as well as to the software testing community, see also in (Balzarotti et al. 2010).

Applying formal methods for e-voting

The usage of formal methods in the specification and verification of e-

voting systems is relatively new. We mention (Kremer and Ryan 2005, Campanelli et al. 2008, Delaune et al. 2009, Sturton et al. 2009), that present formal specification and verification of an voting system at different level of abstractions and practicality. Clearly, these works demonstrate the feasibility of formal verification of voting machine logic, along the line of (Tiella et al. 2006, Villafiorita et al. 2009b, Weldemariam et al. 2009, 2010). Although work in this area is very few, we can group the existing works into two groups according to the level of abstractions and target of evaluation: verifying cryptographic protocols and system behavior (i.e., the control logic of the application software).

Related to the first group, (Kremer and Ryan 2005, Delaune et al. 2009) presented a framework for the specification and verification of three privacy-type e-voting protocol properties. These properties are vote-privacy, receipt-freeness, and coercion-resistance. The authors used applied π-calculus (Abadi and Fournet 2001) to formalize these properties as observational equivalence, after being formalized the voting protocol as a set of processes using the same formalism. In (Campanelli et al. 2008), the authors used a CCS (Calculus of Communicating Systems) like process algebra with cryptographic primitives to specify and analyze some properties of the e-voting system they built. More specifically, they presented a small mobile implementation of an e-voting system named M-SEAS (Mobile Secure E-voting Applet System) and used formal verification technique to validate the security property of the system. Their analysis goal is checking whether their system is free from Sensus vulnerability by using the Crypto-CCS language (Martinelli 2002) and PaMoChSA analysis tool[9].

Related to the second, (Tiella et al. 2006) demonstrated the integration of formal methods in development of the voting system. In particular, the authors specified the behaviors of voting control logic using UML finite

[9] `http://wwwold.iit.cnr.it/staff/fabio.martinelli/pamochsa.htm`

state machine and developed a tool named FSMC+ that automatically generate NuSMV code corresponding to the specified finite state machine. They performed the verification using the NuSMV model checker. Recently, an approach for the design and analysis of an e-voting machine based on combination of formal verification and systematic testing is presented by (Sturton et al. 2009). They formally verify the correctness of each of the individual component of voting machine, as well as verifying some crucial correctness properties of the composition of the components. This work is targeted to the following verification goals: ensuring that each individual component of the voting machine and their composition should meet the specification of the individual components and their composition respectively; voting machine should be structured to enable sound systematic system testing; ensuring that the voting machine must behave and store votes according to the voters selection when configured with a particular election definition file. For each module, they construct a formal specification that fully characterizes the intended behavior of that component. A number of properties related to the structural and functional aspects that the machine should satisfy are identified and specified. They used Verilog (Thomas and Moorby 1991) —a hardware description language for digital circuits, for the implementation of their specification and SMV[10] analysis tool and satisfiability solving (especially, the SMT solver)[11] to verify that their Verilog implementation meets the specifications.

2.3.3 The ProVotE e-voting system

Based on the following archived works (Caporusso et al. 2006, Tiella et al. 2006, Villafiorita et al. 2009b,a, Komminist Weldemariam and Mattioli

[10] `http://www.kenmcmil.com/`

[11] The component-level specifications for some model are difficult to verify using the SMV model checker due to large state space explosion in the SMV model checking to model check at the bit level, the authors proposed to use the SMT solver (Sturton et al. 2009).

2009), we provide an overview of the ProVotE project. The project is sponsored by the Autonomous Province of Trento (Italy) with the goal of evaluating the switch to electronic systems for local elections.

An Overview. Figure 2.3 shows a very high level view of election in Italy and highlights the targets of automation of the ProVotE project. There are two main chains of functions, voters' management (pictured in gray) and votes' management. The former includes all the operations and data for making sure that only the people with a right are given access to voting and that each citizen can cast at most one vote. The latter, instead includes all operations related to managing votes. These two chains of functions, among others, provided various opportunities for automation. In particular, the development of the e-voting and systems to automate the recounting of the ballots produced by the e-voting machine. (The ProVotE e-voting machine is a DRE-VVPAT.)

The ProVotE system is based on various sub-systems that address two main process flows, mostly to implement the two main chains of functions. These are *electronic voting* and *verification* flows. The first includes the electronic devices responsible for automating the voting procedures. Its main sub-systems include *management system* (let say, *configurator*), *voting machine*, and *vote tally system*. 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). The *vote tally system* is responsible for aggregating collected data and publishing provisional results.

The second flow is about verification, that is, verifying electronically produced data. Its main sub-systems include *Paper Ballot Counter* and *Log analyzer*. The first one is a barcode recounting system that helps

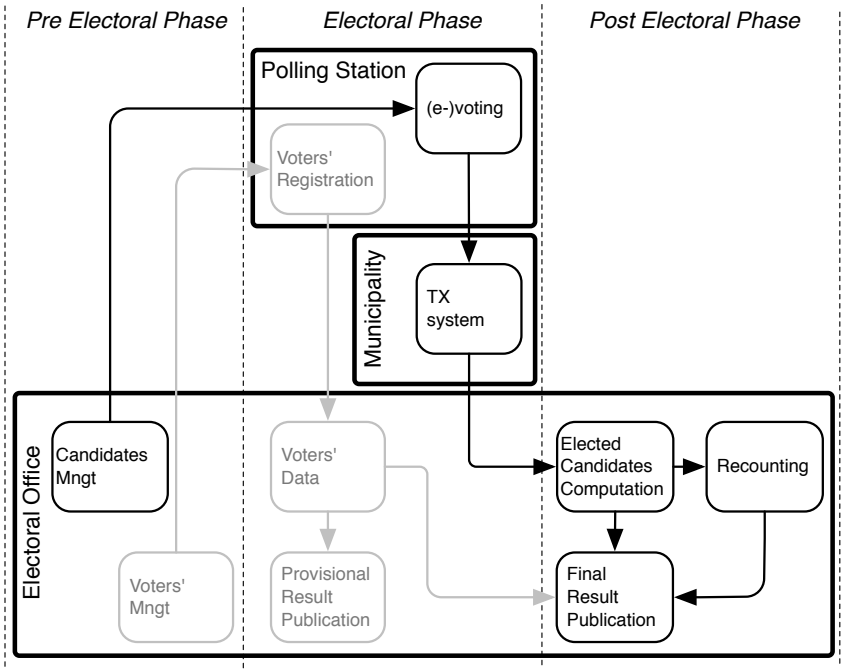


Figure 2.3: Elections in Italy and the ProVotE system architecture. The dotted lines represent the different phases of the voting process: namely, pre-electoral, electoral, and post-electoral. The boxes in bold lines represent the organizations responsible for the process. And, the rounded rectangles represent functions performed during an election.

assisting poll workers, by reading barcodes printed on the ballots. The latter is an an application used, 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* (Villafiorita et al. 2009a).

The ProVotE Development Process. The development of ProVotE has proceeded in cycles, paced by experimentations and with a time-span of about six months each. Each development cycle is a waterfall (Royce 1987), extended and adapted to incorporate BPR, security analysis, and (formal) verification activities. The most interesting aspect of the development is the organization of activities within each cycle, summarized in Figure 2.4.

In particular, the following activities defined the ProVotE development

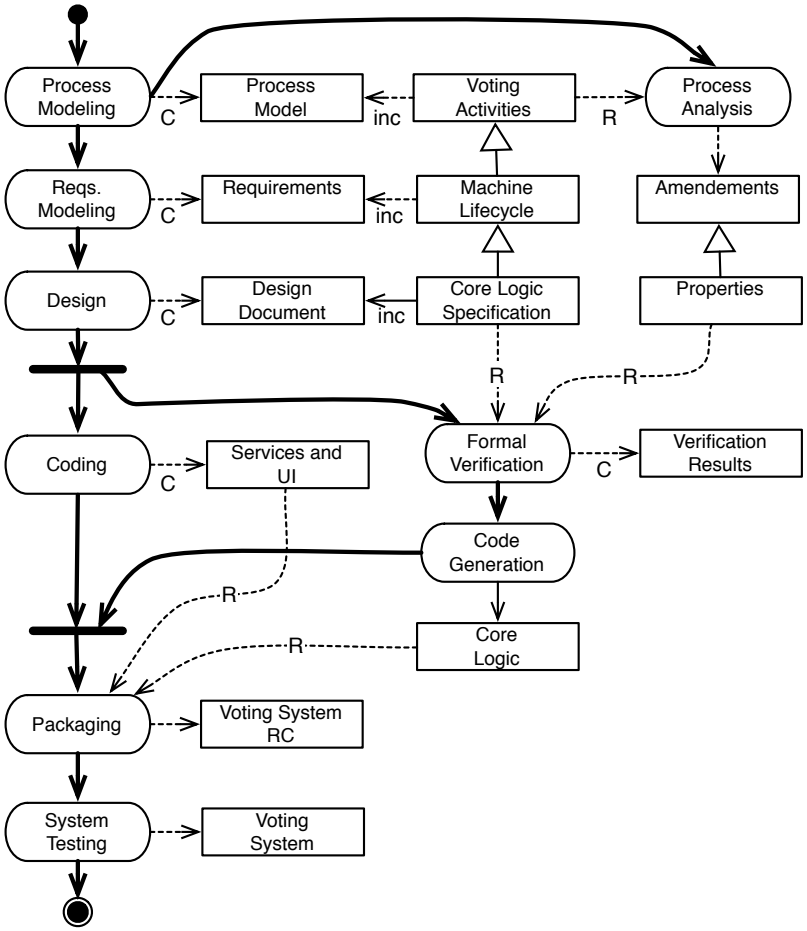


Figure 2.4: The ProVotE Development Process. The figure is an activity diagram in which activities are in rounded rectangles, whose notation are slightly adapted to make it more readable. The meaning of creation ("C"), reading ("R"), and inclusion ("inc"), respectively, are that an activity creates an artifact, it reads it, or that an artifact is contained in another artifact. The empty arrowhead indicates refinement, so that, for instance, the "Machine Life-cycle" refines the "Voting Activities".

process:

- **Process Modeling.** The activity in which process modeling, structuring, and change management are conducted (the concern of Chapter 3);

- **Process Analysis.** The activity in which procedural security is con-

ducted (the concern of Chapter 4);

- **Requirements Modeling.** The activity in which requirements are defined and structured, using high-level statecharts and use cases. The statecharts specify the life-cycle of the machine described earlier and the use cases provide the usage scenarios for each of the states of the machine's life-cycle (e.g., the actions necessary to open the poll site). The voting and the administration user interfaces are specified through a statechart which describes their logic and in which each state corresponds to a different "screenshot" of the system (see the detail in (Villafiorita et al. 2009a));

- **Design.** During the design phase, the statecharts validated by the responsible office have been detailed into *executable* statecharts;

- **Formal Verification.** The *executable* statecharts specification have then been translated into the NuSMV input language, which in turn is formally verified. The translation is done automatically using FSMC+[12] (Tiella et al. 2006);

- **Code Generation.** After having validated the specification of the control logic, FSMC+ has been used to generate the Java code of the *control logic* of the machine (both the administration and voting part). The Java code generated by FSMC+ was inspected by hand, to mitigate risks related to bugs in the translator, and the source code then compiled and packaged to produce the voting application;

- **Coding (and Unit testing).** The code implementing the management of data and hardware devices and some glue code to link the user interface was then implemented and tested using standard practices;

[12] `http://ed.fbk.eu/fsmcp/last/`

- **Packaging and System Testing.** The code generated by hand and that produced by FSMC+ have been finally packaged together and tested using standard techniques.

In summary, the ProVotE e-voting system is quite unique for the following reasons: the development methodology, which is a model-driven integrated development based on UML; the incorporation of formal specification and verification, which is based on the state of the art model checker; the introduction of signaling system, which displays the status of the machine to poll workers such that the poll workers check if the voter has completed voting process correctly or withdrew from voting; the choice of Java and Linux operating system (customized, with specific functionality); the incorporation of users feedback to the development of the machine, that is, information are gathered from various (and different) experiments to get feedback of users experiences on the machine usages and performance of the machine.

Chapter 3

Tool Supported Methodology for BPR

This chapter elaborates an approach where process models for procedures are modeled, and changes in laws are mapped in the models in order to highlight and review the impacts on processes and vice-versa. This allows for a stricter collaboration among the different stakeholders usually involved in BPR. Thus, we first discuss challenges of using BPR for modeling process models in PA and requirements that should be followed and respected across the modeling. We then discuss the modeling methodology, by first presenting the business process model formally and followed by the methodology itself. The methodology guides the identification of the model elements and is supported by a tool named VLPM (Visual Law and Process Modeler). Finally, we demonstrate the approach using various examples taken from electoral procedures and laws in place in Italy.

3.1 Challenges and Requirements of BPR in PA

We discuss the main challenges of the representation, modeling, and analysis of PA processes using BPR. Following that, we discuss the minimum requirements needed for our methodology that allows us to build (business)

process models in a convenient way.

3.1.1 Challenges of BPR in PA

New technologies and the emergence of new paradigms in the relationship between citizens and governments are constantly challenging and questioning the way in which PA delivers its services. This phenomenon is as old as governments are: the Roman Emperor Augustus had among his most relevant reforms an improvement of the PA (Heady 1991). Traditionally, in fact, there are three elements on which governments can operate to improve the way in which they deliver services: *people* (and their *skills*), *processes*, and *technologies*. In Augustus' case, the reform was implemented through people: he transitioned from short-term "amateurs" taken from the privileged class to "paid professionalized civil servants" (Heady 1991). In recent times, interventions always encompass all three dimensions.

A decision in (any) project preliminary phase has more relevant effects that those delayed to the subsequent ones (Sommerville 1995). In the same way, normative choices and changes in PA influence the law effective applicability with respect to the desired system. In principle, we distinguish three different kinds of reengineering projects:

1. *System automation level.* The goal of this kind of project is introducing a new system to better support one elementary task or limited procedure. Typically small in scope, these kinds of projects provide limited improvements but are simple to implement, since they do not affect neither the procedures nor the laws.

2. *Departmental level.* The goal of this kind of project is changing the way in which work is performed within a functional unit, (often) to make it more rational and efficient. These kinds of projects are more impacting, as they require some kind of re-organization of the work,

often accompanied by the introduction of new ICT systems. The impact on the laws, however, is null or minimal.

3. *Inter-departmental level.* The goal of this kind of project is providing a better implementation of those processes that involve different departments or possibly change the allocation of responsibilities or both. It is the case, for instance, of decentralization projects, where competences are moved from central government to districts. These kinds of projects are clearly the most impacting, since they act at all levels, including the normative one.

The first kind of project is a "standard" software development project for which there is a rich choice of tools, development cycles, and project implementation alternatives. However, the other two kinds of projects present two peculiar and closely related challenges, which root is in the relationship between the laws and the processes that implement such laws. These challenges are particularly common in PA domain, which are

- *Laws provide the framework that constrains and limits possible choices and alternatives in reengineering processes.* Providing tools and notations can allow to explicitly model and reason about the alternatives and constraints, and as the same time could help to develop more efficient solutions.

- *Laws and processes are intertwined as requirements and implementation are in software development processes.* Providing tools to explicitly trace the connection between laws and process elements helps for a more efficient and coherent management of the system. This can help ensuring that procedures correctly implement the law, and at the same time it could help to understand which laws might be affected by a change in the processes.

Furthermore, interventions usually require to change part of the law. However, in order to understand where and how to modify the law we have to set, prepare, and validate new processes as well as to recognize the new roles and people responsibilities. Additionally, regulating a complex system or a new one requires to understand about the procedures to activate in order to answer questions like who is in charge of, when the task should be done, how to face exceptional situations, and so on. These all call a significant reform needed to provide correct snapshot about the existing processes and/or procedures, to propose the (re-)design and development of a new system.

One of the tools to enact this reform is the application of BPR techniques. In PA, this is an activity which involves (independently or in collaboration) law-makers who amend laws, process designers who try to optimize existing processes, and software developers, to support existing processes and/or procedures with technology. Modeling facilitates the communication and understanding of the actual organization among these users and is helpful in building a shared vision between domain experts and technicians. Moreover, it provides an easier way of analysis in order to evolve towards efficient and higher quality processes, if not pose related risks.

As discussed in the previous chapter, it is not possible to transfer e-business solutions and development approaches directly to the PA. While most of the existing modeling techniques were developed in order to optimize supply chains or production, there are no such clear goals in modeling public workflows (Alpar and Olbrich 2005). Moreover, unlikely from organizational or firm processes, for instance, the electoral ones are ruled by laws and often several procedures undergo by the experience of a single actor. Thus, on one hand, it is essential to know the constraints established by the law, but, on the other hand, it is also necessary to support the office in charge of such processes in order to decide the changes required

by the new processes. However, as said before, one of the major difficulties encountered in this domain is the strong dependency between processes and laws. Any implementation of software delivery requires a parallel action on both the redesigning of processes and on the introduction of law changes. This means that the current law (in a sense: rather than the processes), must be considered as the constraint, the engine, and the target of the reengineering activity. This link between models and laws raises further issues related to the maintainability of the models over time, since it is necessary to guarantee coherence of the models with the laws in order to have the models retain their value. We argue that the situations can reasonably be tackled by providing homogeneous and structured models of the current processes (the business architecture), which in turn should allow to redesign the new software delivery.

It is important, therefore, to devise methodology and tool that should help tackling the two challenges mentioned above. In fact, we can apply techniques (e.g., goal-oriented methodology) that can help tackling the first problem by providing precise notations and alternatives to avoid mis-interpretation and resolve ambiguities that can arise, and by performing high-level formal reasoning. In contrast, the second challenge can be tackled through a proper BPR approach —namely, by devising process modeling and redesigning methodology and by developing its supporting tool. In this chapter, thus, our focus is in the second direction to address mentioned challenges, on top of Mattioli's work (Mattioli 2006).

We should be clear that our main contribution in this chapter is that of structuring of the previous methodology by providing requirements for the model elements and their formalization. More importantly, this opens opportunity to perform security assessment on the procedures, for instance, "what happens if the procedure is not followed correctly", can be performed.

3.1.2 Requirements for Process Models

The relevant elements that a process model should contain depend on several constraints. We discuss such constraints that are needed to be considered while modeling PA processes. As a minimum requirement, process model should be[1]:

1. *Complete.* A complete process model should contain the relevant subjects, objects, activities, and constraints of PA processes that make up the business architecture. That is, all the significant elements in the PA processes should be mapped to elements of the model;

2. *Simple.* All users and, in particular, users with little or no technical background should be able to read and understand the model. This is essential in order, e.g., to disambiguate the interpretation of certain norms, by using the model as a common language among users;

3. *Formal.* All the elements in the model should be given a precise and unambiguous meaning in order to enable simulations and/or formal verification, if applicable;

4. *Hierarchical.* General (and informal) processes are refined through sub-processes ordered in subsequent levels of abstraction. The hierarchical organization is useful for presentation purposes and to keep the model more comprehensible;

5. *Maintainable.* The organization of the model and of its elements should follow precise and machine-verifiable conventions, in order to guarantee a minimum level of quality and uniformity;

[1] Notice that the requirements listed below represent a complement on the requirements that are listed in (Alpar and Olbrich 2005).

6. *Traceable.* The organization of the model and of its elements should allow synchronization between the model representation and the legal information encoded in the model, in order to guarantee change management between models and such information.

In the first place, models that are constructed for PA processes must be compete —namely, they should contain the relevant subjects, objects, activities, events and constraints of administrative processes that make up a transaction. This includes, e.g., specifying actors who participate in the process and their responsibilities, assets and operations on them, and constraints in the model. Specifying actors (and their responsibilities) in the model not only allows to describe who does what during the execution of a process, but, more importantly in the context of modeling process in PA domain, who manages what data and with what privileges.

As we discussed in the previous chapter, a number of techniques exist for the modeling of business processes. Such techniques and notations must not be too complex since PA users are usually not familiar with modeling languages. This helps avoiding complexity, by maintaining the readability of the models to non-domain users —making the model simple for them but should be complete. At the same time, by creating commonly shared visions among different people about the same process model and by making the models suitable for future revisions. Moreover, although there is a need to analyze PA processes on an abstract level by non-domain users, we also need these techniques and notations to allow lower level analysis by domain experts.

While keeping the models simple and formal, process models should be organized as a hierarchical structure, in order to immediately furnish some generic considerations. It is natural that, on such organization we can distinguish two types of processes: abstract (called “containers”) and concrete (called “executable”) processes. While higher levels in the hier-

archy represent abstract processes, the leaves represent concrete processes. The hierarchical organization helps in browsing the model as well as helps investigating in the real significance of abstract processes in order to reach an agreement on their meaning and split a single complex model into a composition of smaller ones.

In practice, in fact, most people prefer to speak about abstract processes, such as *"voting"* or *"counting"* without knowing the detail about their technical meaning. For a domain expert, for example, *"counting"* has a different meaning if the context is for the local or general elections its main goal is the same but the tasks to achieving it are different. For a citizen, however, it does not matter which task is, as long as the *"counting"* operation just produces the final result. From requirement engineering or software designing point of view, in contrast, *"counting"* for different elections has different requirements, and consequently different implementations.

When we augment a process modeling technique to include information on legal regulations that govern the observed processes, an important aspect is guaranteeing synchronization between the model representation and the legal information, so that any change to the process can be reflected in an equivalent amendment to this information and the other way round. The synchronization of models facilitates their composition with other such processes in order to form a single complete view for their end users, and at the same time by allowing maintaining the structure of the models. The resulting model should be able to show restrictions for the reengineering that are set by the legal framework or other public regulations such as laws, and at the same time by allowing traceability between the models and change management. That is because the improvement of processes depends on the careful analysis of them in which legal constraints must be respected.

Finally, some of the peculiarities of the PA (especially, in e-voting domain) need specific strategies for an effective process modeling, as well as their analysis. The formalization and maintenance of the process models particularly crucial in that regard. Notice that there are limited works or otherwise little said on these two points, namely on formalization of the model and maintenance. Notice also that maintainability of models should not be confused with the synchronization constraint though. Their difference is that the former focuses on the structure of models, in contrast, the latter focuses on the content of the models.

3.2 Representing Laws in XML

We consider a law whose structure is hierarchical organized. In the hierarchy of the law structure, the non-terminal nodes represents structural divisions of the law and terminal nodes are the law paragraphs. Notice that currently, we consider an Italian law as discussed in (Mattioli 2006, Ciaghi et al. 2009a). In other words, for the structure of Italian legislative texts, a law is divided in "Titles" which contain "Articles" that are divided in "Commas" or "Paragraphs"[2]. Law paragraphs contain the actual text of the law and can be further subdivided in "Letters", which are an alphabetically ordered list of clauses. Figure 3.1 shows this structure of (Italian) law representation.

Mark-up languages, and in particular XML, can provide interesting results at both ends of the legislative process: at the drafting stage, enforcing some or all the drafting rules defined for norms; at the accessibility stage, fostering easy and sophisticated searching and rendering tools for the public at large. Furthermore, XML may constitute a great influence on several other aspects of the legislative process, providing support for the consolida-

[2]Respectively "Articles", "Sections" and "Clauses" in the American legal system.

tion of laws, rationalizing the legislative process, improving the referencing and connections among the norms, etc. (Marchetti et al. 2002).

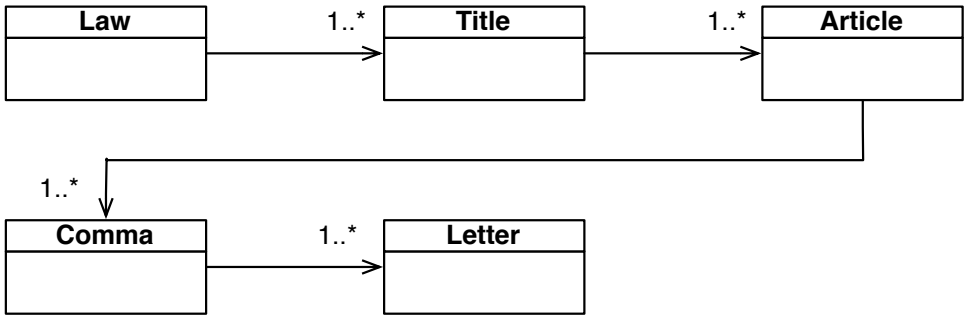


Figure 3.1: Law elements class diagram.

The "Normeinrete"[3] project, started in 1999 by the Italian Ministry of Justice and several other Italian public institutions, defined an XML format for Italian legislative texts that allows the marking of all the elements constituting a law and thus facilitates electronic processing of laws (Lupo 2002, Caterina Lupo 2005). Normeinrete provides several DTDs to support the representation of legal texts. Among these DTDs two are most relevant, namely loose and strict DTDs. The loose DTD supports the former type of texts, while the strict DTD supports and enforces the well-formedness of the latter, namely the adherence to the structure depicted in figure 3.1. Both DTDs describe a structure made of a header, an optional preamble, a hierarchical structure (called *articolato*) containing the elements shown in figure 3.1, a conclusion and a variable number of annexes. These are complemented by the initial and final formulas of the Italian law texts (Marchetti et al. 2002). Our tool and our methodology focus on the *articolato* and therefore we will disregard the other components of the text from now on. In addition "Normeinrete" introduces specific tags to mark resources and entities. This results to be particularly helpful to automate the modeling of responsibilities in processes and extracting a preliminary

[3](Laws on the Net) `http://www.normeinrete.it`

list of used objects.

In order to univocally identify laws and their elements, "Normeinrete" uses a URN based system (Caterina Lupo 2005, de Oliveira Lima et al. 2007). The URNs are defined as a combination of elements according to a specific grammar. The basic elements are: the name of the promulgating authority, type of norm, date, number and a set of more detailed specifications when needed (Caterina Lupo 2005).

3.3 Process Modeling Methodology

Before starting the modeling methodology, first we present formal methodology guidelines with respect to the the consideration we mentioned previously, independently from the UML notations.

3.3.1 Defining Business Models Formally

We introduce business process as a formal and structured model. The model is formal in that it defines a process as a mathematical object, which can be analyzed. The model is structured in that it permits the hierarchical definition of a process, and that hierarchy can be exploited for structuring the modeling and analysis of process models. A mathematical definition for business process descriptions not only enables to obtain their unambiguous interpretation, general, and objective model. It also provides the basis for tool support. Namely, it can contribute to furnish checking mechanisms and to automate the production of some kind of information by respecting constraints imposed by a legal framework. It can also help to be uniform across the modeling phase and prepares the ground to verify if the methodology is correctly applied, thus enabling an efficient development process.

We formally define the business process model as:

Definition 3.3.1 *A process model is a 6-tuple* $\langle A, P, E, R_{AP}, R_{PP}, R_{EP} \rangle$ *where*

- A *is a finite set of actors;*
- P *is a finite set of processes;*
- E *is a finite set of entities (or assets);*
- $R_{AP} \subseteq A \times P$ *is a finite set of actor-process relationships;*
- $R_{PP} \subset P \times P$ *is a finite set of process-process relationships;*
- $R_{EP} = R^2_{EP} \cup R^3_{EP}$ *is a finite set of asset-process relationships where*
  - R^2_{EP} *is the finite set of binary relationships between a process and an entity and,*
  - $R^3_{EP} \subseteq P \times E \times P$ *such that* $(p, e, p') \in R^3_{EP} \rightarrow (e, p), (e, p') \in R^2_{EP}$ *is the finite set of ternary relationships among a source process* p*, an entity* e*, and a target process* p'*.*

An *actor* $\in A$ is responsible for a specific process or asset, and it might be a person, a component, a technical system or a combination of them. Notice that an actor might also be responsible for more than a process. The responsibilities of each actor are realized as a set of roles; and, a role has a name $r \in R$ where R is the set of role identifiers (e.g., *provincial chairperson*, *polling officer*). The assets/entities are what we focus on when performing formal analysis of processes and, in particular, security assessments of the procedures (see Chapter 4 for more detail). Additionally, the R_{EP} is the set obtained from the union of binary and ternary relationships because all relationships involve a process and an entity, but some of them require a more complete description —namely, to model the scenario of sending and receiving an entity (more description later).

In hierarchical organization, processes are distinguished into concrete (or primitive) and abstract (or complex). Usually a process is considered to be concrete if no decomposition will reveal any further information which is of interest. More specifically, we define concrete and abstract processes as:

Definition 3.3.2 *A process p is said to be a leaf or a concrete process iff $\nexists\ p' \in P$ such that $(p, p') \in R_{PP}$. A process p is said to be an abstract process if it is not a leaf process.*

Intuitively, the distinction between concrete and abstract processes is made to distinguish between the fact that a concrete, or a self-defined procedure, is immediately understandable and executable, whereas an abstract process is a non-trivial composition of two or more sub-processes. This distinction allows us to speak about an abstract process to make more general questions, but, at the same time, preserve the opportunity to define it in terms of more refined ones, i.e., sub-processes.

Notation 1

We write $p \sqsupset p'$ if $(p, p') \in R_{PP}$ for some $p, p' \in P$;

We write $p \not\sqsupset p'$ if $(p, p') \notin R_{PP}$ some $p, p' \in P$;

We write $p \sqsupset \ldots \sqsupset p'$ if $\exists p1, \ldots, \exists p_n$ such that $p = p_n \sqsupset p_{n-1} \ldots p_2 \sqsupset p_1 = p'$.

Definition 3.3.3 *Let $p, p \in P$, then*

- *p' is a direct sub-process of p iff $p \sqsupset p'$;*
- *p' is a sub-process of p iff $p \sqsupset \ldots \sqsupset p'$;*
- *Conversely, p is said to be a (direct) super-process of p'.*

To these basic elements, two instruments particularly useful to understand the relationships involving processes, actors, and assets are the RACIV (responsible, accountable, consulted, informed, verified)[4] responsibility and the CRUD (create, read, update, and delete) matrixes, which we borrowed (from project management and database communities respectively) and extended in our modeling. The former points out the relationships among actors and processes in terms of roles undertaken by the different participants actors, whereas the latter points out how the assets are connected to accomplish the process's goal. The name *"stereotypes"* recall the UML profiling technique but the meaning is in a certain way different. We can think them as labels that define the kind of an object. The profiling consists of the following stereotypes sets:

Definition 3.3.4 *The set of all stereotypes are defined as follows:*

- S_A *: a finite set of domain-dependent stereotypes for actors A;*
- S_P *: a finite set of domain-dependent stereotypes for processes P;*
- S_E*: a finite set domain-dependent stereotypes for assets E;*
- $S_{AP} = \{responsible, accountable, consulted, informed, verifies\}$*;*
- $S_{PP} = \{include, extend, generalize\}$*;*
- $S_{EP} = \{create, read, update, delete, send, receive, use\}$

In addition to the standard notation borrowed from the CRUD, the set of S_{EP} is extended by including *use*, *send*, and *receive*. Intuitively, *send* and *receive* labels need a precise asset mobility and underline a change of ownership or control over an asset to be sent and received, respectively.

[4] Also called Responsibility Assignment Matrix (RAM), often used to describe the participation by various roles in completing tasks or deliverables for a project or business process. It is especially important to clarify roles and responsibilities in departmental projects and processes.

They point out that the responsibility or accountability for the involved assets is changing from the accountable actor of the source process to the accountable actor of the destination process. This usually also implies a physical movement of an asset. The main constraint is that suppose we want to model the fact that from a source process $p \in P$, we *send* an asset $e \in E$ to a destination process $p' \in P$. In this case, the relationship $(e, p) \in R^2_{EP}$ specifies the connection between p and e, but it is also true that to understand where e is *sent*. Thus, there exists a corresponding process $p' \in P$ linked to e marked with the *receive* stereotype $(e, p') \in R^2_{EP}$. That is, in fact, why previously we introduced the ternary set R^3_{EP}, that models the dual receive operation. The label *use*, instead, informs the asset is indeed needed to perform an action. It is distinguished from *read* because it does not imply the idea of data or value (legitimate) manipulation. In short, the *use* label suggests that the linked asset is necessary to perform the task but does not provide data by itself.

Example 3.3.5 *The domain-dependent stereotypes for S_P and S_E, for the case of electoral system can contain the following:*

- $S_P = \{$*local, general, single polling station, multiple polling stations*$\}$;
- $S_E = \{$*document, program, object(physical), file*$\}$.

The methodology suggests the above base stereotypes but other domain-dependent ones can be added in the set.

Definition 3.3.6 *Let the codomain of s_i does not contain empty set, $\forall i \in \{$A,P,AP,EP$\}$. Then, we define the following functions:*

- $s_A : A \to 2^{S_A}$;
- $s_P : P \to 2^{S_P}$;
- $s_E : E \to S_E$;

- $s_{AP} : R_{AP} \rightarrow 2^{S_{AP}}$;
- $s_{PP} : R_{PP} \rightarrow S_{PP}$;
- $s_{EP} : R^2_{EP} \rightarrow 2^{S_{EP}}$.

Now consider the law structure we discussed in the previous section, which is organized hierarchical where non-terminal nodes represents structural divisions of the law and terminal nodes are the law paragraphs.

Definition 3.3.7 *Let $LTree_i$ the tree of the law i and let LP_i the set of paragraphs contained in $LTree_i$, then we define $L_i : P \rightarrow 2^{LP_i}$, where if $p \in P$ is defined in $LTree_i$ then $\emptyset \neq Li(p) \in 2^{LP_i}$, whereas if p is undefined $L_i(p) = \emptyset$.*

In the above definition, the meaning of "$p \in P$ is defined" is that the process p is described in $LTree_i$ and $L_i(p)$ is exactly the set of paragraphs involved in its description.

Additionally, the models should respect certain constraints in order to be homogeneous in representing, structuring, and composing them. A set of constraints related to actors, processes, assets, and relationships can be used to encode restrictions imposed on the models. Constraints can be formally expressed like before, by using the same machinery.

Notation 2

Let $x \in S_{AP}$. We write x(a,p) $\Leftrightarrow$ $x \in s_{AP}(a, p)$for some $a \in A$,$p \in P$.
Let $x \in S_{PP}$. We write x(p,p) $\Leftrightarrow$ $x = s_{PP}(p, p')$ for some $p, p' \in P$.
Let $x \in S_{EP}$. We write x(e,p) $\Leftrightarrow$ $x \in s_{EP}(e, p)$ for some $e \in E, p \in P$.

Some constraints for S_{AP}, S_{PP}, S_{EP}, and L_i

The models should respect the following constraints.

Constraint 1 *All actors and assets should be connected to relevant processes. That is,* $\forall a \in A.\ \exists p \in P$ *such that* $(a, p) \in R_{AP}$ *and* $\forall e \in E.\ \exists p \in P$ *such that* $(e, p) \in R_{EP}$*;*

Constraint 2 *The actor-process, process-process, and asset-process relationships should have their own stereotype. That is, the definitions* s_{AP}*,* s_{PP}*, and* s_{EP} *are total functions on* R_{AP}*,* R_{PP}*, and* R_{EP} *respectively;*

Constraint 3 *For each process* $p \in P$ *in the process tree, there exists at most an* $\ll$*accountable*$\gg$ *actor. That is,* $\forall a, a' \in A,\ \forall p \in P$ *such that* $accountable(a, p) \wedge accountable(a', p) \rightarrow a = a'$*;*

Constraint 4 *Only leaf processes can have* $\ll$*responsible*$\gg$ *actors. That is,* $\forall a \in A,\ \forall p \in P$ *such that* $(a, p) \in R_{AP} \wedge responsible(a, p) \rightarrow p$ *is a leaf process, that is., executable;*

Constraint 5 *All leaf processes should have* $\ll$*responsible*$\gg$ *actors. That is, for each leaf process* $p \in P$*,* $\exists a \in A$ *such that* $(a, p) \in R_{AP} \wedge responsible(a, p)$*;*

Constraint 6 *A sub-process contains a subset of the parent process stereotypes in* R_{PP}*, i.e.,* $\forall p, p' \in P$ *such that* $p \sqsupset p' \rightarrow s_P(p') \subseteq s_P(p)$*;*

Constraint 7 *Relations with stereotype* $\ll$*include*$\gg$ *in* R_{PP} *means the subprocess has the same set of the parent stereotypes, i.e.,* $\forall p, p' \in P$ *such that* $p \sqsupset p' \wedge include(p, p') \rightarrow s_P(p') = s_P(p)$*;*

Constraint 8 *Relations with stereotype* $\ll$*generalize*$\gg$ *in* R_{PP} *means the subprocess has a proper subset of the parent stereotypes, i.e.,* $\forall p, p' \in P$ *such that* $p \sqsupset p' \wedge generalize(p, p') \rightarrow s_P(p') \subset s_P(p)$*;*

Constraint 9 *Relations with stereotype* $\ll$*generalize*$\gg$ *split the super-process' stereotypes in a collectively exhaustive way, i.e.,* $\forall p, p' \in P$ *such that* $p \sqsupset p' \wedge generalize(p, p')$ *then* $\exists p_1 \ldots \exists p_n\ \forall i = 1 \ldots n,\ p_i \neq p \wedge p \sqsupset p_i \wedge s_P(p_i) \neq s_P(p') \wedge s_P(p) = s_P(p') \bigcup_{i=1 \ldots n} S_P(p_i)$

Constraint 10 *For each asset-process in* R_{EP} *relationship whose stereotype is* $\ll$*send*$\gg$ *there exists an asset-process in* R_{EP} *relationship whose stereotype is* $\ll$*receive*$\gg$ *and the two processes are connected in* R^3_{EP}*. Let* $e \in E, p, p' \in P$ *such that*

$(p, e, p') \in R^3_{EP} \rightarrow send(e, p) \wedge receive(e, p')$

$(e, p) \in R^2_{EP} \wedge send(e, p) \rightarrow \exists p' \in P$ *such that* $(p, e, p') \in R^3_{EP}$

$(e, p') \in R^2_{EP} \wedge receive(e, p') \rightarrow \exists p \in P$ *such that* $(p, e, p') \in R^3_{EP}$*;*

Constraint 11 *Each sub-process has a subset of law paragraphs of its super-process, i.e.,* $\forall p, p' \in P$ *such that* $p \sqsupset p' \rightarrow L_i(p') \subseteq L_i(p)$ *for each law* i*;*

Constraint 12 *The law of the super-process contains the union of the law of its sub-processes. Let* $p_1, \ldots, p_n \in P$ *be all the direct sub-processes of* $p \in P$*, then* $L_i(p) \supseteq \bigcup_{i=1\ldots n} L_i(p_i)$*.*

The next section shows how we use the formal model of the business process in the modeling methodology. Namely, we show the usage of the formal definition for the UML models by following the methodology.

3.3.2 The Modeling Methodology

The XML representation of laws provides several advantages among which eliminating (some) syntactic ambiguities and providing formal and automated cross-references. However, when the target of the law is the definition of the operational aspects of the PA, i.e. how the PA works, and the target of the work is the re-engineering of processes, workflows and workflows diagrams are more commonly used. As the size of the model increases, issues concerning composition, uniformity (i.e., the same notation is used coherently in all models) and quality of models written by different people quickly arise.

For this reason, a modeling methodology along with modeling guidelines which are meant to support functional analysts in defining their processes is presented in (Mattioli 2006) and later used in (Ciaghi et al. 2009a). The requirements and formal model discussed previously are the basis for defining the methodology, which is the added value on top of the original methodology as proposed by Mattioli. The methodology is organized in the following main phases:

- *Preparation phase.* During which data and structure for the modeling elements are identified and organized;
- *Static process modeling phase.* During which a hierarchical structure of processes is constructed and actors are linked to processes in the hierarchy independently from processes' execution and their analysis;
- *Dynamic modeling of processes.* During which we describe the processes workflow by emphasizing sequential and parallel activities.

The approach is based on UML notations mainly because software engineers are familiar with it. Some kinds of diagrams are easily understood by domain experts who are not conversant with software development and it is supported by several CASE tools. Specifically, the methodology uses the following UML diagrams: *use case diagrams*, which are employed to provide the static hierarchical process organization allowing to concentrate on processes and actors with their responsibilities; *activity diagrams*, which are used to deal with the dynamic aspects of the system, rendering processes as actions which transform assets and their state; UML *actor* and *object* node represent an actor $a \in A$ and an asset $e \in E$, respectively; and *package* and class *diagrams* are used to represent meta level concepts.

Actors, use cases, and object nodes can be stereotyped with elements of S_A, S_P, and S_E respectively. The law paragraphs in LP_i are represented by means of instance specifications. Notice that the approach is not strictly bound to UML diagrams. Similar diagrams and notations can be used as long as they allow to represent some concepts included in the methodology (e.g., multiplicities) and support tool extension. Additionally, the two instruments (i.e., RACIV and the extended CRUD) discussed previously about the relationships involving processes, actors, and assets are used.

Notice that the notation basically conforms to the approaches proposed in the past. The added value is a set of rules and conventions that simplify

maintenance and allow to more closely match the notation to some peculiarities of the electoral procedures and laws and the use of a precise and simple methodology based on the UML notation and formal definitions. In what following, we discuss each phase of the methodology in detail, using the mentioned UML notations to concretize its usage and emphasize its generality.

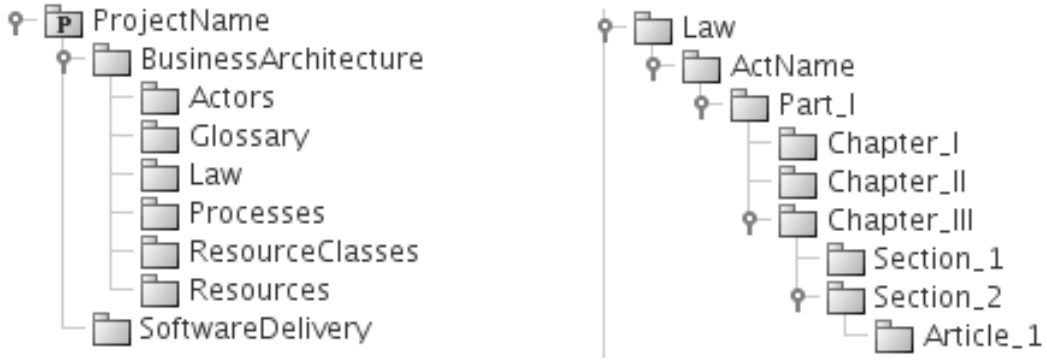
3.3.2.1 Preparation Phase

In this phase of the methodology, we identify and prepare the data and structure for the process model using packages and classes diagrams. The phase mainly comprises of the following activities:

- **Actors and Assets Identification.** For what concerns actors and assets identification, we further split this activity into:
  - **Actors identification:** aims at extracting roles, actors (humans or organizations) and groups (e.g., offices) which are involved in the domain under modeling and analysis and their connections with other actors;
  - **Asset classes identification:** types of the assets involved and their relationship with processes and other assets;
  - **Assets identification:** instance of assets types which are needed to perform the different operations. This phase requires an understanding of the evolution in *state* and *value* of the assets —i.e., how they change after executing a process and if they become less or more critic. These information, in fact, are the basis for procedural security analysis.
- **Stereotypes identification:** the domain specific characterizing aspects and concepts are identified and agreed among all parties. Namely,

the sets S_i are constructed, $\forall i \in A, P, E, AP, PP, EP$. This allows to understand to which extent the methodology fits to specific problem and to assess the extension points.

- **Terminology identification:** the domain specific glossary is collected and agreed among all parties.
- **Laws collection:** in this activity we collect and enumerate laws which rule or influence the domain under analysis.



(a) Structure of packages.

(b) The law sub-package.

Figure 3.2: Package structure.

Figure 3.2(a) shows the main concepts organized in different packages after the preparation phase. The *law package* organizes the laws regulating the processes. Each law has its own sub-package which is hierarchically structured in other sub-packages. Each sub-package is a non-terminal law element (e.g., a chapter or a section of the law) while leaf elements like law paragraphs are represented by means of instance specifications. Each package may contain an *object* diagram modeling its internal structure and the instance specifications. Title and description (e.g., the text of the law) are attached to both terminal and non-terminal law elements. In contrary, Figure 3.2(b) sketches the *law package* structure of an act and its possible subdivision.

The *actor package* contains all UML actors representing roles or organizations which can contribute in the realization of some processes. It can include one or more *use case* diagrams describing the relationships among them. Each actor is described including what the actor represents (office, person, role, etc), its functions and responsibilities. One or more use case diagrams underline relationships among actors. Possible relationships among actors are *generalization* and *association*.

The *resource class package*, *classes* for short, holds all the asset types used as input or output by some processes (see also Figure 4.2). A class is the abstract definition of an asset related to a process, while the asset itself is an instance of that class lowered in a specific context. One or more class diagrams describe the relationships among classes using standard UML relationships. This is meant to provide better characterization through dependency, aggregation/composition, and generalization relations. Similar to UML specification, a dependency relationship is used whenever an object defined by a class depends on another class. Collecting all the instances of the assets involved in some process flow is responsibility of the *resource package*. It mainly contains UML ObjectNodes whose type will be one of the classes defined in the *resource classes* package. During this activity, moreover, useful characteristics about each asset can be extracted, which in turn are also the basis for performing procedural security analysis.

Finally, the *glossary package* includes the domain specific terms used in the model. They can be represented by UML classes, i.e. each word or acronym is described by means of the UML class.

3.3.2.2 Modeling Static Perspective of Processes

At this point, we have enough information to construct the static view of the processes (what) together with the actors (who). The static view of the processes focuses on recognizing attributes that are independent from

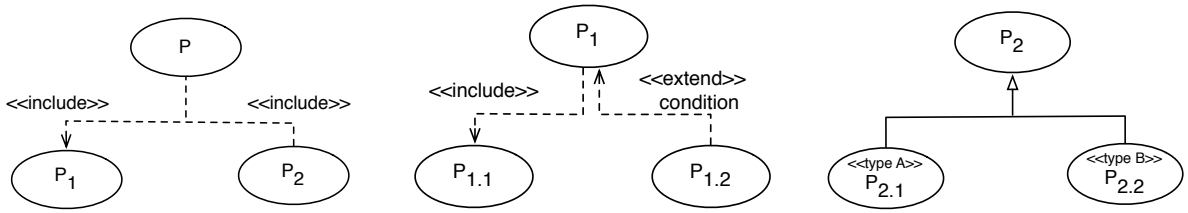
their execution. UML use case diagrams are employed to elaborate this perspective. We recall that the static perspective of processes can either be derived from domain experts or from a law describing some procedures, as well as if the law is documented in some form such as in XML.

The main activities in this phase include *process breakdown*, *association* of actors and their responsibility to processes in the processes structure, and *law association* with process models (see in Section 3.3.2.4). In the process breakdown activity, using top-down analysis, processes are organized as closely as possible into a tree (but not necessarily) structure by preforming one-to-many decompositions. Following this, the responsible actors are associated to processes in the process hierarchy. Additionally, law paragraphs are defined and linked to processes in the process breakdown structure.

Process Breakdown

As said previously, in the process structure represented by use cases, the higher levels represent *abstract* processes while the leaves represent *executable* processes or *tasks*. Figure 3.3(a) a generic root process $p \in P$ is decomposed in two sub-processes $p_1 \in P$ and $p_2 \in P$, i.e., $p \sqsupset p_1$ and $p \sqsupset p_2$. This is also called first *level of the hierarchy*. Sub-processes p_1 and p_2 can then be further defined using one use case diagram each and by decomposing it in order to obtain two new decomposition levels. If p can be decomposed into p_1 and p_1 in $p_{1.1}$ ($p_{1.1}$ is a sub-process of p, $p \sqsupset \ldots \sqsupset p_{1.1}$), then two different use case diagrams should be used.

Processes can be colored, labeling them with one or more stereotypes. This results useful to create group of processes which go across fixed decomposition hierarchy in order to recognize similar types of processes inside the domain. To make an example, local elections and general elections are



(a) A decomposition level. (b) Decomposition types. (c) Generalization.

Figure 3.3: Static view: relationships between a super-process and its sub-processes using UML.

similar enough[5] to be modeled together, but election specific processes can be marked according to their election's type when differentiation is needed using ≪*local*≫ or ≪*general*≫ stereotypes, for instance. This is useful in the subsequent development process such as to structure and maintain requirements, as we demonstrated in (Villafiorita et al. 2009a).

Another important point for the static view of the process structure is the relationships among processes, i.e., S_{PP}. In particular, the S_{PP} relationships are identified by three relationships types: *decomposition*, *exceptional behavior*, and *generalization*. We represent a normal decomposition using the ≪include≫ stereotype relationship, by respecting also Constrain 7. Its meaning is that the sub-process is needed for the realization of its super-process and always executed. The second kind of relationship, exceptional behavior, is mapped to the ≪extend≫ relationship. In this case, the sub-processes are executed only if certain circumstances are verified, for example, if a special condition holds, which can be specified on the connection itself.

Figure 3.3(b) shows these two types of relationships. To show the fact that, for instance, special procedures are required such as voters who require assistance to vote due to some physical impairment or inability, the ≪extend≫ stereotype can be used. Generalization, in contrast, occurs

[5]As far as Italian elections are concerned.

when an abstract process needs to be refined according to a specific instance (Figure 3.3(c)). The tally of results, for instance, can be thought as an abstract process to apply the counting procedure. In that case, it has to be refined according to the specific election type (e.g., a referendum or an election), each one related with particular counting algorithms.

One methodological constraint for modeling with UML is to produce for each abstract process $p \in P$ a use case diagram and an activity diagram (see Section 3.3.2.3) in order to show only the relations with its direct sub-processes. This helps to keep the diagrams simple and to focus on one process at a time. In addition, to better illustrate the hierarchical structure we suggest to put the abstract process on the top of the diagram and its sub-process under it from left to right in chronological order.

Additionally, start and end timing constraints (or other information to include in the process description) can then also be attached to each leaf process to explicitly define when they are supposed to be executed. In UML, this can be carried out by means of *tagged* values.

Actors and Responsibilities

The next step in modeling processes from the static point of view is the specification of participating and responsible actors (multiplicity). This information describes who does what during the execution of a process.

After actors have been identified, it is possible to extend the hierarchical process structure with actors' information. Figure 3.4 represents a decomposition level where *Actor A* is accountable for the super-process and two *Actor B*s are responsible for one of its sub-processes. Notice that the UML notations is flexible enough to allow the specification of the number of participating actors. There are several examples where this information is useful, for example, when two witnesses are required to sign a document. The relationships between actors and processes can be labelled according

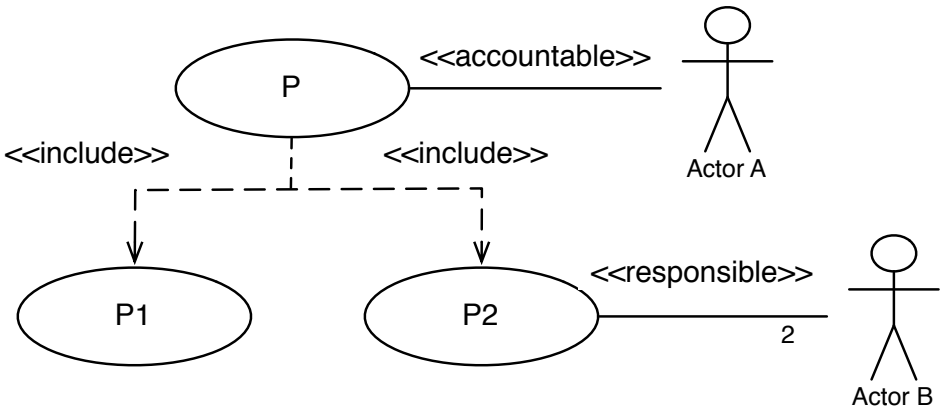


Figure 3.4: Some RACIV relationships among processes and actors.

to the set R_{AP} (i.e., the RACIV matrix) and some implicit assumptions can be made to respect the constraints discussed earlier (in particular, Constraints 3 and 4). For example, when an actor is accountable for an abstract process, it is implicitly accountable for all of its sub-processes if not otherwise specified, or s/he is also responsible for a sub-process if no other responsible actor is explicitly stated.

3.3.2.3 Modeling Dynamic Perspective of Processes

In this phase of the methodology, each *entity-process* relationship $(e, p) \in R_{EP}$ is mapped to activity diagrams. These are used to describe how direct sub-processes are composed to generate the direct super-process. Hence, each abstract process has to have its activity diagram that describes the composition of its sub-processes. Furthermore, the activity diagrams describe the processes workflow by emphasizing sequential and/or parallel activities, using the triggering conditions identified in step one.

More specifically, given a *use case* model that represents the process decomposition of a process $p \in P$, the corresponding activity diagram is built in the following way. All the sub-processes of p are modeled as actions with the same name of the sub-processes and connected together

using the activity diagram notation in order to describe the process flow. In the diagram, assets are connected to the actions and each relationship $(e, p) \in R_{EP}$ is labelled by a subset of the stereotypes in S_{EP}. Note that the ≪send≫ and ≪receive≫ stereotypes both in S_{EP} can be added in the activity diagram only if there is a triple $(p, e, p') \in R^3_{ER}$ for $p, p' \in P$ and $e \in E$, as remarked by the Constraints 10.

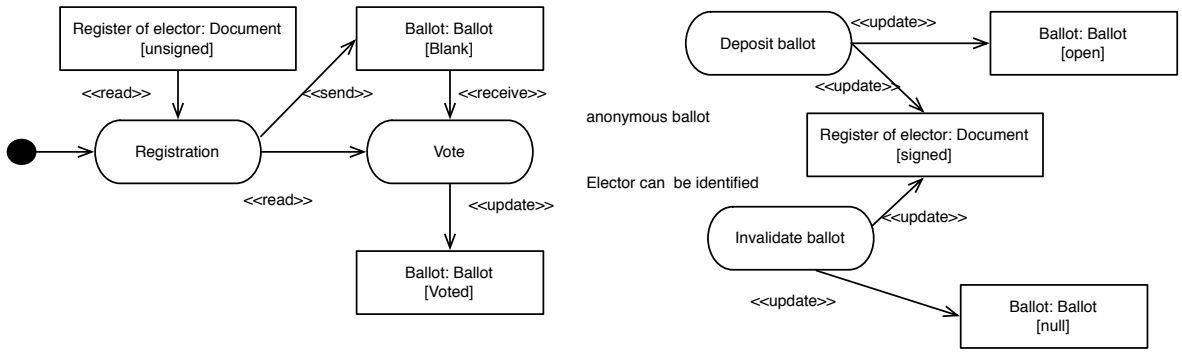


Figure 3.5: Activity diagram example.

Figure 3.5 shows a simple example of activity diagram. It models the fact that the registration of voters is checked by polling officers to know if the voter is eligible to vote. If so, the responsibility of the *blank* ballot is then transferred to the voter who votes and changes the *state* (and value) of the ballot. After that, either the ballot is anonymous and cast or it is recognizable (e.g., there is anything written to identify the voter) and hence invalidated. If not otherwise specified, the input asset is considered unchanged after execution. For example, the first process *reads* the register of voters leaving it unchanged while the following ones *updates* the register (e.g., the voter has voted). This allows to explicitly specify what action is performed on what assets and how the asset changes its state.

Stereotypes can also be applied on the assets themselves according to S_E to highlight crosswise concepts. The methodology suggests the base stereotypes shown in Example 3.3.5.

3.3.2.4 Linking Laws and Models

In this activity, we associate laws with processes in the process models. More specifically, if the static perspective of processes are derived from a law describing some procedures, then it is necessary to trace a relationship between the process and the law. This eventually provides a strategy to keep the model up to date with the law and in order to determine how BPR interventions affect the original law. More specifically, a general UML dependency can be added between the process and the law element (e.g., a law paragraph) contained in the *law package*. Namely, to associate a subset of law paragraphs $L_i(p)$ for the process $p \in P$ in the "process tree", each *use case* is linked with an *object* diagram —i.e., specified as packages and instance specifications.

An example is shown in Figure 3.6. Note that the model is fully documented since all its elements contain their relevant description (extracted from the text of the law or written by the domain experts). It is not necessary to explicitly draw the relationship among processes and law divisions. In fact, "Normeinrete" URNs can be used to link the process model with the text of the law, since URNs univocally identify laws and their elements. This choice is compliant with the current (Italian) law editing standards.

Maintaining law-model traceability allows to automatically identify which parts of the law should be amended by tracing back to the parts of the law that originally defined the modified processes Because assigning dependencies between processes and laws is a laborious task, this activity is mainly intended to be tool supported and will be discussed more in the next section. We remark that this activity can be conducted before the modeling of the dynamic perspectives of the process model using activity diagrams, as discussed in Section 3.3.2.3. As a first iteration, we suggest to follow this order. This is because, the static view of the process model

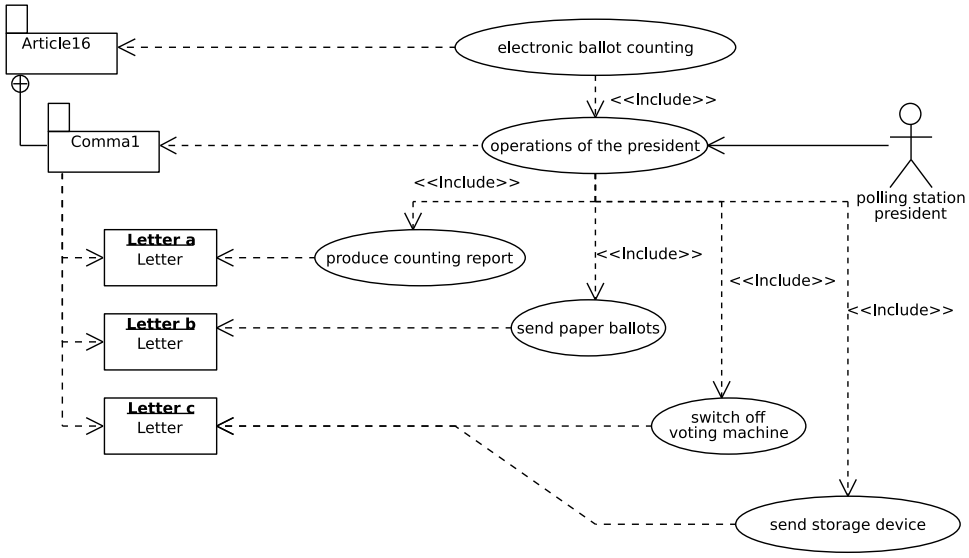


Figure 3.6: Example of an article and its defined processes.

is the basis for dynamic view, which eventually becomes the starting point for the analysis of procedures —i.e., the procedural security analysis, a topic of the next chapter.

3.4 A Tool for Supporting the Methodology

In order to support the methodology presented previously and to facilitate the translation of law into a process model as well as the maintenance of both when they change, a tool named VLPM (Visual Law and Process Modeler) was developed. The tool is built on top of Visual Paradigm for UML[6] and it is freely available from `http://ict4g.fbk.eu/vlpm/`. In the following, we describe the tool internal representation and its usage scenario.

[6] Visual Paradigm is a commercial UML modeling `http://www.visual-paradigm.com/`

3.4.1 Intermediate Representations

Figure 3.7 shows a high-level representation of the model elements, i.e. a *metamodel* for the tool. The diagram mainly shows the internal representation of the model elements. In the diagram, a process is realized as an observable activity executed by one or more *actors*. Actors can be extracted from the text of the law or can be defined manually. Currently, our model identifies them by means of an unambiguous identifier (extracted from the XML file containing the law information or manually specified) and a name. However, this could easily be extended in order to add more features, for instance, stereotypes. In the same way as actors, assets can be either extracted from the law or defined manually. If the assets are extracted from the law, we then store their initial states in the model and use our notation to define the changes that the assets undergo.

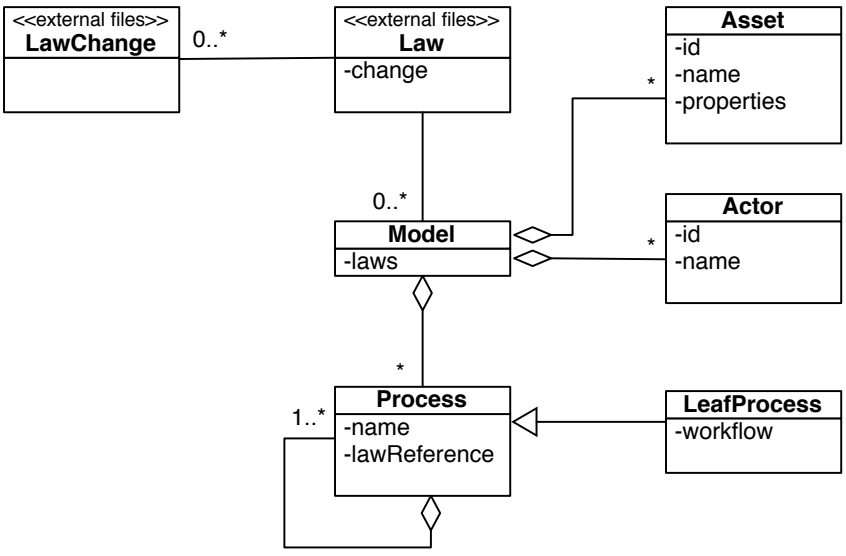


Figure 3.7: The internal representation of our modeling elements.

In addition to the modeling elements, we use a generic relationship elements to create specific sub-classes of relationship as shown in Figure 3.8, which are defined separately from the elements of the model. *Actor-Actor* relationships have different properties from *Actor-Process* relation-

ships (e.g., the allowed stereotypes) and from *Process-Process* (R_{PP}) relationships. The association of a process with its executing actors (i.e., *Actor-Process*, R_{AP}, relationship) is based on the static assignments of the responsibilities (set of roles, R) to the actors. This information can be extracted from the law or manually assigned after the actors have been identified. The use of an abstract relationship object allows us to create as many types of relationship as we need, with the only requirement of defining also a suitable translation of each relationship to UML. The model also explicitly support the *Asset-Process* (R_{EP}) relationships that define the semantics for the asset flows.

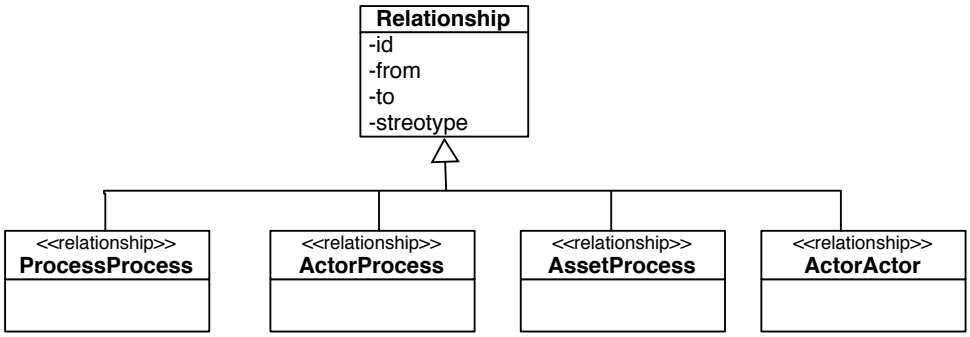


Figure 3.8: Relationships among the modeling elements.

The model represents the static information of the business processes, while the dynamic properties (namely, asset transformation functions) are defined in a specific notation. The model is associated to the laws that regulate its business processes to allow the association of a single process with relevant law parts that define them. Notice that the law is not included in the model. We use examples from the Italian law system retrieved using the Normeinrete project and its XML format. Although our model is designed to support XML format for laws representation, it can be easily extended to support other formats.

3.4.2 VLPM Usage Scenario

Typical VLPM usage scenarios are depicted in Figure 3.9. The flow of the modeling activity is organized in the following steps:

1. A law written in natural language is marked with XML tags;

2. The user imports the law formatted in XML and VLPM generates a skeleton of the model. The user needs to verify and complete the generated model in order to have a reliable *as-is* view (i.e., a "process-tree" view) of the law. This model can be exported in various formats for documentation purposes;

3. The user imports an Explicit Text Amendment that modifies the law that has been previously modeled with VLPM. The tool highlights the impacts of the amendment on the law and on the model, allowing the user to focus on the affected parts of the model. This greatly simplifies the model revision process;

4. The user modifies the process model, re-engineering some processes. At this point documentation can be generated to be shared among the stakeholders and to compare the as-is and the to-be models. Moreover, VLPM can be used to generate the XML skeleton of a new law that amends the originally modeled law.

As noted above, the process of modeling a law with VLPM begins with the generation of a UML model from a law described in XML, namely by importing the law. Notice that not all the elements of a law can be translated into business processes. Thus, a human intervention is needed to choose which law elements are relevant. However, the tool allows the user to choose which law elements will be part of the UML model. The import procedure has been implemented as follows:

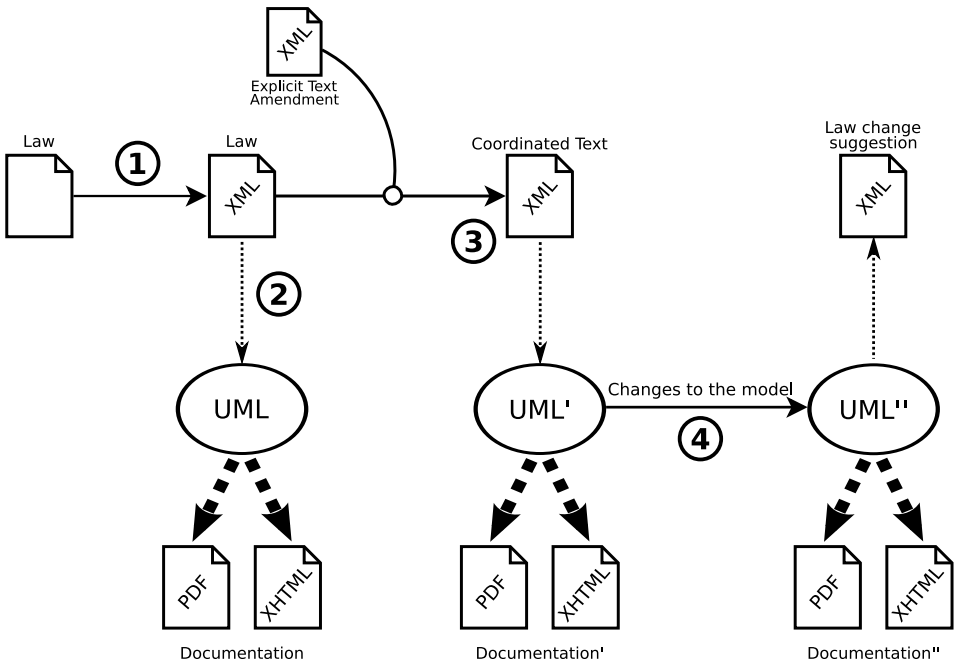


Figure 3.9: VLPM usage scenario, i.e., Law modeling process handled by the VLPM tool.

First, the XML file containing the law is parsed and a business process is automatically generated and associated to each element of the law tree. Actors are also identified and associated to their relevant process by looking for XML tags that mark assets. Second, a GUI displays the tree of the law elements, their content, the associated actors and processes. This interface allows the user to choose which actors and which processes to include in the UML model. By choosing a subset of actors and processes, it is possible to study only a specific aspect of the law (e.g., all the operations of which a certain subject is responsible). Moreover, by displaying the content of each law element, the user can decide if it is meaningful to associate a process to that law element. Finally, once the user is satisfied with the model layout, VLPM generates the UML model containing all the elements of the law and a tree of the processes. Thus, the model generated is supported by a serialized data structure that links it to the text of the law in order to maintain traceability (see Figure 3.10). In the figure, the intermediate

representation (i.e., *Intermediate Data Structure*) is the one described in the metamodel above.

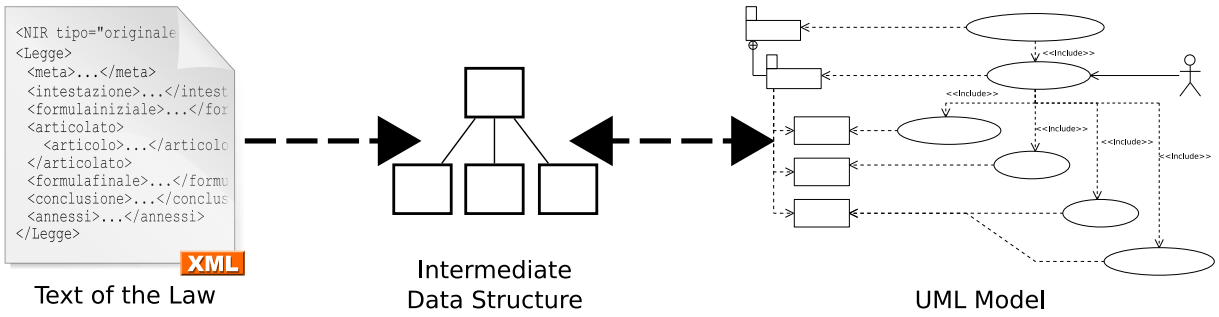


Figure 3.10: Storage of Law Data.

VLPM also allows to import *explicit text amendment* (in Italian "Modifica Testuale Esplicita", also called MTE). This is a law that modifies a part or the whole text of a "comma" or a "letter" or changes the structure of a part of another law. The Italian law system allows several types of changes to a law, the most important of which is called *Novella*. A *Novella* introduces new elements in a law, by substituting some other element(s). The current implementation of VLPM focuses on this kind of modification.

When importing a "Novella", the tool behaves similarly to the case in which a new law is imported. However, the parser looks for XML tags that mark a modification. VLPM then allows the user to review the effect of the "Novella" on the law that is being modeled. In the case of a simple text change, they can decide how and if the processes and the relationships with the involved actors should be modified in the model. In the case of a MTE containing a structural change, the user is provided with the same interface of the law import phase but can intervene only on the part that is being modified.

When a law amendment is promulgated, it exists in parallel with the original law. For our purposes, we need to have a *Coordinated Text* containing the most updated version of the law. Thus, when the change is

applied, our data structure is updated so that the text it contains is modified by the amendment. From that moment onwards, the traceability will be maintained with the *Coordinated Text* instead of the text of the original law.

Change Management using VLPM

In order to perform the change management on the models, VLPM adds two specific functions: *model refinement* and *Law change suggestions*. By providing these two functionalities, VLPM allows a law-model-law *round-trip*, facilitating the law-making process in the case of regulations for PA procedures. More specifically, the former allows to maintain traceability with the text of the law using specific functions. These functions are meant to be used to give a better representation of the procedure defined by the law and not to add or remove procedures from the actual regulation. The latter, provides suggestions based on the new processes that cannot be traced to any part of the text of the law. When a new process is added to the model, VLPM generates a list of suggestions that can be used to produce an *Explicit Text Amendment* (in "Normeinrete" XML format) from the changes undergone by the model, thus allowing the law to be realigned to the model.

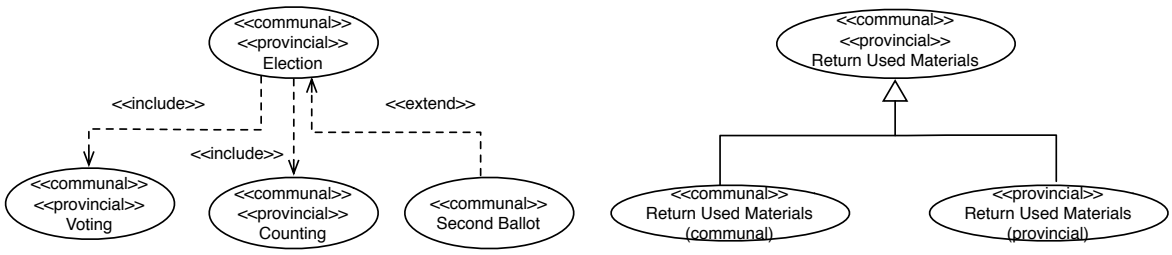
We need to be clear that as of the current version, the tool only supports the Italian laws system. However, work is in progress to make the tool more flexible and more functional in various areas, among which will be the support for different XML representations of laws (which are used by VLPM for linking process and laws); more flexibility in deployment (e.g., by allowing integration with freely available UML tools); integration with formal analysis techniques for simulation and verification. We are also working on applying the approach for other legal domain by combining with goal-oriented methodology (Villafiorita et al. 2010).

3.4.3 Examples

The organization of elections is a complex public administration (PA) process. It spans over a time period of months, it involves several actors and various PA offices, and it has a rich set of enforcing rules. Electoral laws precisely define all the steps that have to be performed to run fair elections. Failure to comply with the law may result in litigations in the simplest case and in a threat to citizens' fundamental rights in the worst case. In the scope of the ProVotE project (see the previous chapter), in line with these all, one critical aspect is defining the new electoral processes and delivering a complete solution compliant with the national and local laws. The methodology and tool have been applied in two real case studies with in the ProVotE project. Specifically, for the modeling of the electoral law and procedures in the Province of Trento and the law for the introduction of an e-voting system for a local poll in two municipalities of the Autonomous Region of Friuli Venezia Giulia.

With respect to the first case study, the methodology was adopted to model town and provincial elections in the Province of Trento (Italy), producing diagrams which describe about 80 processes, 30 actors and over 90 entities, while the two main reference laws include from 80 to 100 articles each. The VLPM documentation features have been applied to make the model available to all the stakeholders. The methodology, along with the tool was subsequently used to re-organize voting processes in order to support the ProVotE e-voting system development (Caporusso et al. 2006, Villafiorita et al. 2009b).

Figure 3.11 shows a simple example of processes' relationships R_{PP} taken from the Province of Trento electoral models. Figure 3.11(a) means the process named *"Election"* is decomposed in two sub-processes: *"Voting"* and *"Counting"*. The third process, *"Second Ballot"* is linked with the



(a) Include and Extend relationships.

(b) Generalize relationship.

Figure 3.11: Relationships among processes.

≪extend≫ relationship because it is optional and not always necessary to complete an election. In Figure 3.11(b) the process *"Return Used Material"* is common in both communal and provincial elections, but actors involved, sub-processes or mechanisms are different according to the election type (see also Constraints 8 and 9).

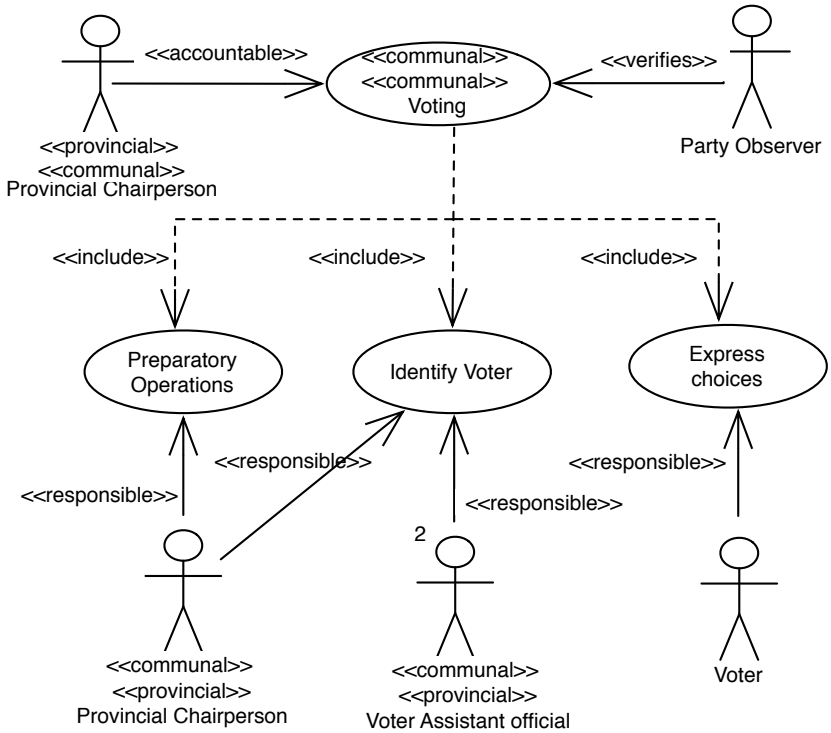


Figure 3.12: A simplified use case diagram for the voting process with its direct subprocesses.

In Figure 3.11(a), we already saw the first decomposition step for the *"Election"* process. Figure 3.12 shows the second decomposition level where the *"Voting"* process is considered. The actor with a role *"provincial chairperson"* is *"accountable"* for all the three sub-processes, besides his/her accountability to *"Voting"* process, since this process is the abstract process of the three processes. Moreover, the process *"Identify Voter"* on which there are two actors that contribute to perform this task and one of them has the multiplicity set.

With respect to the second case study, VLPM was applied to rule the introduction of e-voting in a small municipal referendum in Friuli Venezia-Giulia. The referendum involved about six-hundred people, who used e-voting system with legal value. In the following we take an example of the law from Friuli Venezia-Giulia case and show its modeling and analysis using the VLPM. The law we consider was published on 27 July 2007 and became effective on 8 August 2007. The original law is made of 21 articles, organized in 3 Titles. The first step of our scenario was to add XML tags to the original text, in order to make it compliant with the *complete* "Normeinrete" DTD. Before importing the original law it was necessary to instruct VLPM to slightly change the default structure automatically proposed both to disable law paragraphs which do not define any process and to associate more processes to the same law paragraph.

As an example, article 17, Comma 2, Letter `a` defines three atomic and clearly distinguishable procedures: "load data", "data aggregation", and "voters number verification". Using VLPM it is possible to associate to letter `a` three processes representing the three procedures. However, Title 1 contains only general principles and rules regarding referendums that do not define any specific procedure. Thus, they are not needed in the processes model and VLPM can be instructed not to generate processes out of this Title. The law was then imported and used to create a UML

process model, following the choices previously described. This model was then used for printing the documentation and facilitating discussion among the stakeholders of the project by providing a visual representation of the processes.

As the election came closer and some constraints became more evident, a "Novella" was then written, containing some changes for the original law. In general, the following change types can be recognized. The first type is related to changes to the text of the law which do not influence the structure of the processes. An example of this is Article 7, Comma 2, where the "Novella" modified a time constraint. Instead of requiring the installation of the voting machines in polling station some day before the election day, the new text requires them to be installed the day immediately before the election day. Despite this being only a case study, this change is realistic. The reason for this change could be the need for more security in order to avoid leaving the voting machine exposed to tampering after its installation at the polling station.

The second is related to those changes in the relationship between actors and processes. Finally, structural changes such as the subdivision of a paragraph in letters, each one defining a particular sub-process. An example of structural modification is in Article 17, where Comma 4 was introduced. The comma was divided into three letters that specify three atomic sub-procedures of the activities that the Regional Electoral Service must perform at the end of an election. The result of this modification is that the process associated to Comma 4 has now three new sub-processes.

The model refinement functions can be used to split the leaf processes that do not describe elementary procedures. For example, Article 15, Comma 1 specifies three operations and is by default associated with only one process. It is reasonable to split the process associated to it in three separate processes (see figure 3.13). This course of action is more compliant

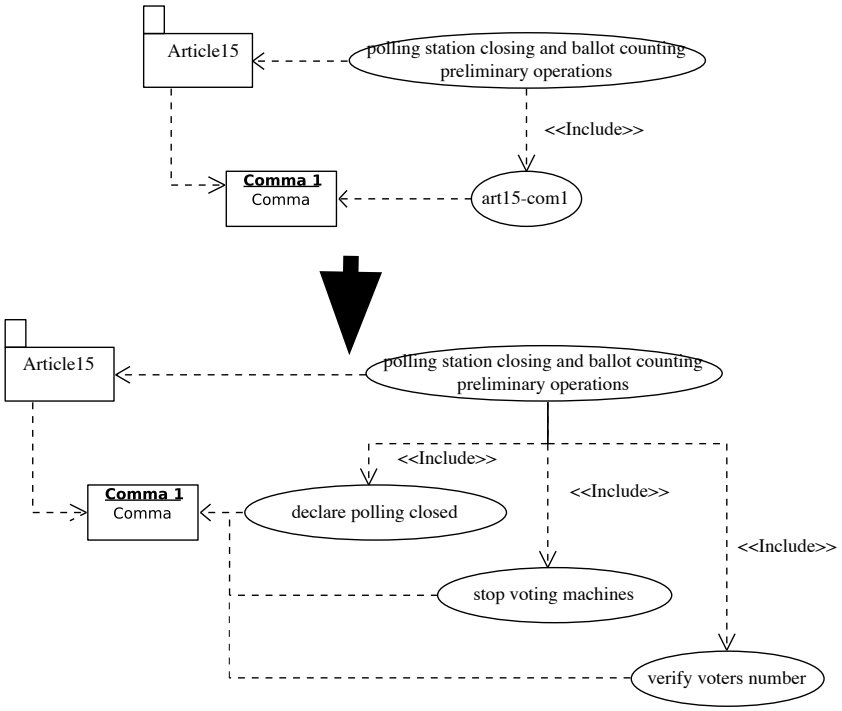


Figure 3.13: Example of the *split* function applied to Article 15, Comma 1.

to our methodology and allows a more rigorous analysis of the procedures associated to Comma 1.

Finally, by having a look at Article 11, Comma 3, we discovered that it does not specify in a sufficiently detailed way the suitability check of a voting machine. Since this is a crucial issue for both user confidence and actual security, it is reasonable to envisage a more specific set of operations and thus create new processes that define the sequence of operations required to verify the suitability of a voting machine (or to define more stringent constraints). In this case, VLPM identifies the new processes as "orphans" and suggests a "Novella" containing a structural modification in order to include them in the law. Such a template can then be "translated" in a more specific language.

3.5 Summary

In this chapter, we have discussed a method for (business) process modeling in public administrations. We presented the requirements, modeling elements, and some formalization on them. We then discussed our tool supported methodology. It comprises of preparation of the data and structures which are presented using *package* and *class* diagrams, modeling the static perspectives of the process model (using *use case* diagrams), and modeling the dynamic perspectives of the processes (using *activity* diagrams). Finally, we discussed the VLPM tool and its application with case studies.

For what concerns the evaluation of model presentation and comprehensibility, the methodology allows to organize system processes in a hierarchical way (as "process tree") and attach information with it in order to answer which are the system processes, who is accountable for them, which resources are used, how they are related, and so on. Moreover, the activity diagrams describe the processes workflow emphasizing sequential and parallel activities, whose assets are needed and how their state evolve, i.e. how they change after execution of a process.

Using the methodology, it is straightforward to link the laws and models in order to increase the traceability between them. One of the goals of traceability is helping law makers elaborate models in collaboration with software developers or process engineers, and understand the impact of law or process changes. This helps, first, to justify the existence of a particular process by providing a reference to the parts of the law that define it, which in turn allows us to link the process to all the constraints in the law that regulate it. Secondly, it allows to understand the impact of a change both in the law and in the process model. When a change is made to the law, being able to identify which processes are defined

(or regulated) by the modified part of the law allows us to modify the process model accordingly. By looking at the model, it is then possible to determine what processes "interact" with the processes affected by the change in the law. The modification can then be propagated to all the relevant processes and makes the model up to date. On the other hand, the reengineering of processes may result in a need to modify some parts of the law. Maintaining law-model traceability allows to automatically identify which parts of the law should be amended by tracing back to the parts of the law that originally defined the modified processes.

It is worth emphasizing that, although our description is in the context of electoral domain, the methodology and the tool can be customized for other domains. To that matter, we applied the methodology to model and analyze the Italian Immigration law in (Ciaghi et al. 2009b). Namely, the entrance procedures at the Italian border and the required documents as defined in the 286^{th} legislative decree of July, 25 1998 on "Consolidated Act of Provisions concerning immigration and the condition of third country nationals" and in the 394^{th} presidential decree of August, 31 1999 (with all their changes until now).

Chapter 4

Procedural Security Analysis

This chapter presents a systematic approach for analyzing processes and critical assets that hold sensitive information from the point of view of security. Our goal is understanding how the switch to the new technology changes risks with the ultimate goal of defining the laws and the procedures regulating system process, that guarantee a higher level of security. We introduce what we call procedural security and devise repeatable methodology to perform the analysis.

Thus, first we discuss the reason why we need procedural security and the benefit it brings on top of technical security. Second, we present the conceptual framework for the approach we propose. Third, we present the methodology for procedural security analysis, by first presenting the formal model for the procedures under analysis and followed by the methodology itself. Finally, we demonstrate the approach using examples taken from ProVotE e-voting system.

4.1 Why Procedural Security?

We all take actions to avoid security risks in our daily life. It may be as simple as putting security guards or locking the office door when leaving for a day up to applying sophisticated security mechanisms. For our

home computer, maybe it is sufficient to enforce good security policies and procedures such as activating the firewall and keeping the OS updated with relevant security patches. The situation becomes more complex and difficult if the system is a major information system that provides critical services through several procedures for the (digital) society. This is exactly the case of (voting and) e-voting in which the procedures are complex in understanding and implementing. Even in those countries that have adopted a high level of automation, the executions of procedures and controls, carried out by people on (physical) assets (e.g., printouts of the digital votes), remains a necessary and unavoidable part.

Procedures in voting system are best practices mandated locally or at the national level. These practices are intended to ensure that a particular election is carried out correctly and securely. Thus, they are often as important as the technical security features of election systems, since they elaborate and transform critical assets. In many cases, where the purpose of the procedure is not apparent to the individual performing it and doing something else is more convenient, it may be true that it is not often at all. Any procedure, no matter how well crafted should be viewed at best an imperfect mitigation, and at worst a mere suggestion. In light of this, those setting procedures should carefully consider what happens when a procedure is not followed, for example, once, occasionally, or frequently.

Interestingly, paper voting and the procedures regulating paper elections are not "immune" to attacks, which can usually be carried out under the hypothesis of multiple "failures". For instance, if a ballot is stolen before the election and the polling officers do not report it, it is possible to control how voters vote in a particular polling station. Attacks on the paper system certainly do happen, but the distributed nature of paper elections limits their scope (McGaley 2008). In order to affect an election without being blatant, one would have to spread any attack over many ballot boxes. Each

new ballot box requires new member of the conspiracy. The introduction of new technologies in the polling stations changes the risks and attacks that can compromise or invalidate an election. On the contrary, a successful attack on e-voting system could be implemented on a wide scale by altering the "right" asset (e.g., the voting software). The feasibility of such risks and attacks not only depend upon the security level the new systems provide, but also on the procedures and controls regulating the way in which the systems are operated.

In order to ensure a sufficient level of security for systems, therefore, there is a need for a thorough security risk analysis methodology that considers procedures as part of the modeling and the analysis process. The approaches discussed, so far, e.g. as discussed in the state of the art chapter, on security (risk) assessment said less or otherwise not effective in *procedurally rich* systems. By procedurally rich systems, we mean that situations in which software systems are just part of a more complex organizational setting and in which procedures have to be executed on security-critical assets that belong both to the digital and physical realms.

Procedural security deals with the identification, modeling, establishment, and enforcement of security policies about the procedures that regulate the usage of a system and system processes. The breach of security objectives during the execution of the procedures and indirectly to systems and system processes, is known as *threat to the procedures* (or *procedural threats*). We call *procedural security analysis* the process of understanding the impact and effects of procedural threats, namely courses of actions that can take place during the execution of the procedures, and which are meant to alter, in an unlawful way, the assets manipulated by procedures.

The situation (reference scenario) of procedural security analysis is shown in Figure 4.1. In the figure, the shaded actors represent a set of adversaries, whereas the non-shaded represent trusted actors. Our target of evaluation

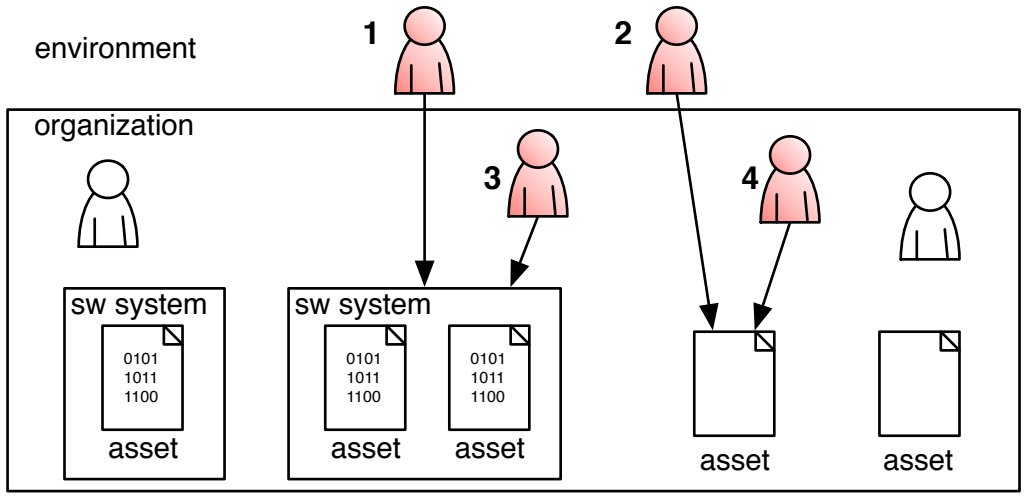


Figure 4.1: A reference scenario for procedural security analysis

is a (complex) organizational setting in which procedures transform and elaborate assets. Assets may change over time while executing procedures. The procedures and organization are meant to add value to the assets and to protect them from attacks, which can either come from external sources or from insiders.

In particular, we distinguish the following kinds of attacks:

1. *Attacks on digital assets* (item 1 and item 3 in Figure 4.1). These kinds of attacks are meant to alter one or more of the digital assets of an organization. Attacks can either be carried out from external sources (the environment) or from internal sources. Opportunities for attacks are determined by the organizational setting and by the security provided by the digital systems.

2. *Attacks on other kind of assets* (item 2 and item 4 in Figure 4.1). These attacks are meant to alter one or more of the non-digital assets of an organization. Attacks can either be carried out from external sources (the environment) or from internal sources or a combination of both that forms coordinated attacks. Opportunities for attacks are

determined by the organizational settings only. The attacks may lead to compromise digital assets as well (e.g., stealing a password)

Security assessment like (Fovino and Masera 2006, Hogganvik 2007) usually focuses on understanding items 1 and 3, namely, types and effects of attacks on (software) systems. We propose a tool-supported methodology to tackle also points 2 and 4 above, namely types and effects of attacks on assets that are not necessarily digital and that derive from the way in which procedures are implemented and carried out.

Technical elements of the approach. In order to achieve the stated goal, we approach the problem by reasoning about the procedures and controls that regulate the usage of system we call *procedural security analysis*. We do so, by proposing a customizable methodology, which is summarized as follows:

- **Step 1.** *Provide models of the procedures under evaluation.* During this step we provide models that describe the procedure or procedures to be analyzed. In order to ease the task of translating the models into executable assets flows, we stick to a subset of the UML notations.

- **Step 2.** *Extend Model.* During this step we generate *extended model* from the models defined in the previous step. The extended model is generated by injecting[1] threat actions into the nominal flow of the procedures. Thus, in the extended model, not only assets are modified according to what the procedures define, but they can also be transformed by the (random) execution of one or more threat actions.

- **Step 3.** *Encode the Asset Flows.* During this step we derive assets-flow in terms of executable specifications —by using the NuSMV input

[1] Note that by fault injection we mean the extension of the assets-flow model with a specification of the possible threat-actions. We adopt this terminology, which is standard in the safety analysis, even though it may not be fully appropriate (Bozzano and Villafiorita 2007).

language, for the model obtained in the previous step. The NuSMV model of the asset flows is based on the definition of program counters that ensure that procedures are executed according to the specifications, and by defining one module per asset with one state variable per asset feature. The state variables encode how features change during the execution of the procedures. Accessory information, such as actors responsible for the different activities can be used, e.g., to enrich the language used to express security properties. The necessity of modeling actors roles in NuSMV depends upon the target of the security analysis.

- **Step 4.** *Specify Security Properties to Model Check.* During this step we specify the (un-)desired (procedural) security properties — namely, the security goals that have to be satisfied (unsatisfied), are then encoded using LTL/CTL formulas, which in turn together with the model are given as input to the NuSMV analysis tool.

- **Step 5.** *Perform Analysis and Results Analysis.* During this step we run the model checker to perform analyses. If a property is proved to be false, the NuSMV tool generates a counterexample which opens up further discussion. Counterexamples of security properties encode the sequence of actions that have to be executed in order to carry out an attack on an asset.

Using this approach analyzing the attacks becomes similar to a model checking problem in which the required final state of some key assets is expressed using LTL/CTL and the counterexample generated by executing the *extended model* contains the sequence of threat-actions causing the final state not to be reached. Additionally, the model checker will take care of pruning useless threats, namely threats which do not lead to any successful attack. Analogously to what happens in safety analysis when

analyzing, e.g., the loss of critical functions, enhancing the procedures results in reducing the probability of an attack or making the attack more complex, rather than eliminating it (Manian et al. 1998, Hsiung et al. 2007, Bozzano and Villafiorita 2007).

In particular, performing such analysis (i.e., by analyzing the counterexamples) has the following two benefits:

1. it helps to identify the security boundaries. That is, the conditions under which procedures can be carried out securely. More specifically, using the NuSMV model checking facility, it is possible to understand what are the hypotheses and conditions under which a given security goal is achieved or breached.

2. it helps to devise a set of requirement, to be applied both at the organizational level and on the (software) systems used to make systems and system processes secure. This can be achieved by analyzing the generated counterexamples by the NuSMV analysis tool, since counterexamples provide information to try and modify the existing procedures so that security breaches are taken care of.

In the rest of the chapter, we focus on each technical element in detail. Subsequently, we use a subset of UML diagrams in order to highlight and describe the domain concepts in a strict and defined way. These include information about workflows, assets, and actors and their role when participating to different workflow activities. Moreover, the diagrams help to make easier the task of translating the models into executable specification.

4.2 Conceptual Framework

Before performing the modeling and formal analysis, we first need to recognize the core elements of the model under analysis. These information

include assets, processes[2], and attacks. The abstraction of these facilitates the reasoning of some properties of interest, such as the lifecycle of assets.

4.2.1 Framework to Understand an Asset

An important aspect of a methodology is "how" we characterize assets, which contain all the sensitive information. This requires to understand and then distinguish their structural and behavioral characteristics. Namely, how we describe assets and their evolutions in a systematic way so that interesting formal analysis about (un-)desired behaviors of the assets can be carried on.

Figure 4.2 represents the main concepts we wish to model in order to perform procedural security analysis. The starting point is a set of processes, that are performed by actors with different roles (left hand side of the figure). The explicit representation of actors is an important feature of the model. In fact, it allows us to represent security breaches related to opportunities for insiders (e.g., an actor participating in different activities has access to all the resources needed to carry out an attack; an actor responsible of a security-critical operation has no supervision) and the security boundaries of the procedures (e.g., how many actors do I need to involve to carry out at attack?).

In the figure, a process can have zero or more inputs, representing the information that is, under some conditions, required for the execution of the process. It can have zero or more number of outputs, the information that the process provides to the next level. Processes use and transform assets, which, in our representation, can either be primitive or containers, namely, assets which may contain other assets. There can be zero or more number of preconditions, which must all hold in order for the process to successfully apply the transformation. The process can have one or more

[2]Throughout this chapter, a process and an activity (or workflow activity) can be used interchangeably.

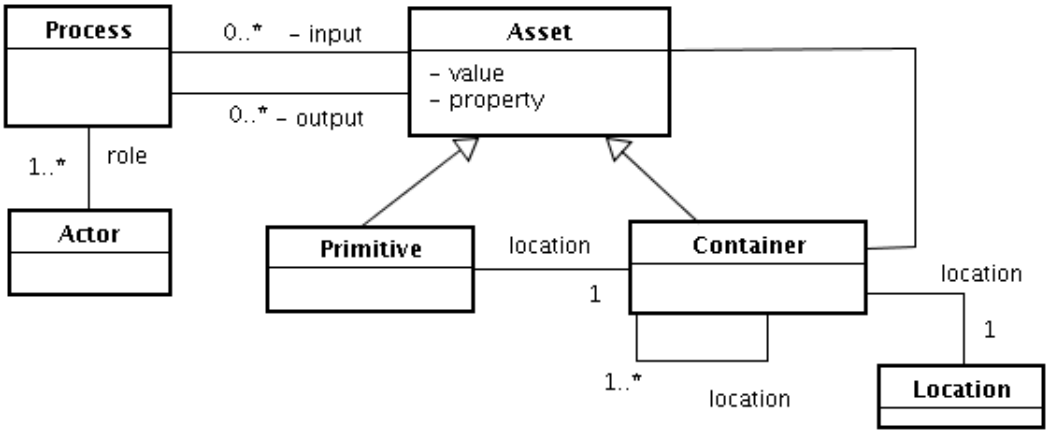


Figure 4.2: A class diagram depicting the characteristics of an asset.

number of effects, in terms of state change over the input assets. Outputs and effects can depend on conditions that hold true at the time performing the process. We call features of an asset all its structural characteristics. The features of an asset and the way in which such features are changed by the processes correspond to the behavioral aspect. This provides a lot of information to understand possible weaknesses of the procedures and the impact of attacks, for instance. It should be clear that in the previous chapter we only indicated the elements of the process models irrespective of the security modeling and analysis.

For the analysis goals we wish to carry out, in particular, we characterize assets with the following features: nature, value, location, content, a set of security specific properties, and number of instances.

The *nature* of an asset can be *primitive* (such as names, symbols, keywords, passwords, electronic ballots, and electronic data in general) or *containers*, when they contain other asset or a set of assets (e.g., a memory support that can contain electronic ballots). It is immutable, namely it cannot be changed during the execution of the procedures. The *value* of an asset (which we represent qualitatively, e.g., *no value*, *high value*, etc),

allows to reason about the impact of threats (e.g., an attack to an asset with *no value* does not cause any harm). The initial value of *primitive* assets is assigned in the model by a domain analyst, and upon the execution of an activity it may change. The value of container assets is determined by their intrinsic value (determined by the analyst) together with the value of the assets they contain.

In our methodology, assets are situated in a *location* (e.g., an electoral office, a safe, a container asset). Asset might be placed in several locations as well as being in transition between various locations. Breaking into a location or being able to access a container is a mean to lead an attack against a (contained) asset (e.g., stealing a memory support containing electronic ballots allows to attack the electronic ballots and can cause a discrepancy among the election outcomes). The initial locations are determined by the analyst and can be changed by the execution of an activity. The location of primitive assets corresponds to the location of the containers in which they reside. Furthermore, a container asset can be a relative location for another container assets as well. This allows us to speak about which kind of (sensitive) information an asset has at a certain (discrete) point in time, which we call *content* of an asset. The *content* of an asset can be known in advance or the execution of a workflow activity over the asset assigns a content or set of contents to the asset —e.g., an *envelop* that is used to ship the voting kits to polling station may contain all relevant information needed to run the election, such as the election software, PIN, etc.

We allow to characterize an asset also by means of a set of *properties* that describe the current situation of the asset such as the security measures that are enacted (e.g., a safe can be *closed*, a file can be stored in *plain* or *encrypted*). While for any domain there can be domain-specific states of the asset, we are particularly interested in security states such as *open,*

closed, unsigned, signed, encrypted, plain, etc.

In addition to the mentioned qualities, a number of copies (called *number of instances*) is particularly relevant in the electoral domain because the same asset can be replicated many times. When printing the ballots, for instance, copies go through different responsible people. This is also important to estimate the effects of realizing a threat against an asset before or after a duplicating operation.

Finally, we call *state* of an asset the values of the features of an asset (i.e., value, location, number of instances, and property) at a given instance, and we call *asset-flow* the sequence of states through which an asset goes during the execution of a process. Even though an asset can have multiple instances, we focus on how a single asset instance can evolve from some initial state through other states as the result of workflow activity executions. An example: the abstract concept of *ballot box* (class) is associated with several instances which describe its evolution during processes execution, for example a *state* change from "open" (e.g., during the voting phase) to "closed & sealed" (e.g., right after the election is over). Also its *value* changes accordingly, which is "low" when the *ballot box* is empty (before the election) but which becomes "very high" after voter deposits a marked ballot.

4.2.2 Asset Threats and Attacks

The information reported previously (along with the previous chapter) is sufficient to describe non-trivial flows of the assets in terms of state transition systems in the *nominal* case —i.e, when all the assets flow as expected. However, this is not always the case because the number of attacks continue threatening the nominal flow of systems and/or procedures either for financial gain or political interest. Therefore, we need to identify all the possible cases in which assets can be threatened through a proper security

analysis methodology.

The key step in security analysis is *"how"* we describe the attack model (Bishop 2002, Xu and Nygard 2005). It describes the intents a malicious actor might have (e.g., modify the election software before the election, sabotage election result), the types of malicious actors (e.g., election officials, poll workers, messengers, or voters), the privilege to each type (of malicious) actors (e.g., who has read/write access to a critical asset), and the location where malicious actors can possibly implement the attack.

For our purpose, it is enough to assume Figure 4.1 as a reference model that distills the essential characteristics of a procedural rich scenario. As noted in the figure, we highlighted two types of assets (digital and physical assets), the attackers' nature (from insiders or outsiders perspective), and the environment in which the procedure transforms the assets under analysis. In principle, an attacker might wish to pursue any kind of goals with different attack dimension (see below). We consider the following goals of an attacker: altering the procedure in unlawful manner to compromise data integrity (e.g., produce incorrect vote counts), compromising the availability of a system causing a denial of service attack (e.g., block some or all voters from voting in key polling site, delay the announcement of election results), compromising the confidentiality of the system (e.g., allow voters to cast doubt on the legitimacy of the election results).

4.2.2.1 Attack Dimensions

It is important to give different dimensions of attacks, since it allows to define the sequence of actions that can potentially lead to a particular attack under consideration. Which attacks are feasible depend on the attacker capabilities and types of attacks on the assets (on the system, in general). Not all attacks, in fact, result in equal damage when they considered. Looking at the consequences associated to systems and/or

procedures breach, thus, is a good way to categorize different dimensions of attacks.

Along the lines of (Kohno et al. 2004, Balzarotti et al. 2008), we can categorize attacks along the following dimensions, mostly for e-voting systems: *detectable vs undetectable*, *recoverable vs unrecoverable*, and *preventable vs detectable.*

In the first case (detectable vs undetectable attacks), some attacks are undetectable no matter what practices are used. In contrast, others are detectable in principle but are unlikely to be detected by the routine practices currently in place. The potential harm caused by both classes of attacks surpasses what one might expect by estimating their likelihood of occurrence. For instance, the mere existence of vulnerabilities that make likely-to-be-undetected attacks possible casts doubt on the validity of an election. If an election system is subject to such attacks, then we can never be certain that the election results were not corrupted by undetected tampering. This opens every election up to question and undercuts the finality and perceived trustworthiness of elections. Undetected attacks are a potential menace to the democratic process: think for, e.g., undetected alteration of the electoral results in one or more polling station. Therefore, we consider undetectable or likely-to-be-undetected attacks to be especially severe and especially high priority while performing the analysis.

In the second case (recoverable vs unrecoverable attacks), in some cases, if an attack is detected, there is an easy way to recover. In contrast, other attacks can be detected, but there may be no good recovery strategy, e.g. short of holding a new election. In intermediate cases, recovery may be possible but expensive. For example, recovery strategies that involve a full manual recount impose a heavy administrative and financial burden and will introduce delays in the finalization of the election results. Unfortunately, this could probably take back to the known attacks in paper

voting. Even if errors are detected and corrected, the failure again has the potential to diminish public confidence. At the same time, detectable but not recoverable attacks are arguably not as serious as undetectable attacks: we can presume that most elections will not be subject to attack, and the ability to verify that any particular election was not attacked is valuable.

In the third case (prevention vs detection attacks), there is a trade-off between different strategies for dealing with attacks. One strategy is to design mechanisms to prevent the attack entirely, closing the vulnerability and rendering attack impossible. When prevention is not possible or too costly, an attractive alternative strategy is to design mechanisms to detect attacks and recover from them. Most election systems combine both strategies, using prevention as the first line of defense along with detection as a fallback in case the preventive barrier is breached. This combination can provide a robust defense against attack.

Attack dimensions are further distinguished between wholesale vs retail and casual vs sophisticated, according to the damage they can cause with the level of granularity to implement the attacks themselves.

4.2.2.2 Asset Threats and Attacks

By assigning each asset value and location we can highlight in a model where a threat may be implemented and how much harm it can cause. A denial of service, for instance, could be caused by deleting a valuable asset (an asset whose *content* value is not null) when the asset is in a given location. The notation used in the previous chapter, however, does not allow to represent the how, e.g., the flow of actions causing termination of the procedures in some undesired state nor to reason about composition of threats —that is, what happens if multiple assets are attacked simultaneously. To tackle this, we propose the concepts of *asset threats* and *attacks*. The reference scenario and the attack dimensions discussed earlier

help defining these concepts in the following way.

We define a threat (or threat-action) to an asset as an action that alters feature(s) of the assets or allows some actors privileges (e.g., a "read" privilege) on some assets. That is, it is what an adversary might try to do to an asset in a system and it is described by a sentence like *"adversary does something to the asset"*. Like workflow activities, threat-actions can transform assets or their state or both. Unlike the workflow activities, however, they transform assets to non-nominal situation, that is, the assets are in undesired states due to the effect of the threat-action.

To differentiate between the nominal and non-nominal behavior of assets, we enlarge the set of processes and assets by distinguishing between:

- *malicious processes*: we identify some of the possible operations that an adversary may conduct against an asset, and
- *malicious assets*: we describe all the instruments and data built or obtained by an adversary to achieve his/her goal(s).

Malicious processes can require as input some particular assets, produce new assets to complete other wicked activities, and modify the original assets to alter the asset flow. To give an example, a sniffer could be considered as a malicious asset needed by the adversary for capturing the password and the result is a new malicious asset: the unauthorized password copy in plain to use for other purposes. This further led us to distinguish between threat-actions into:

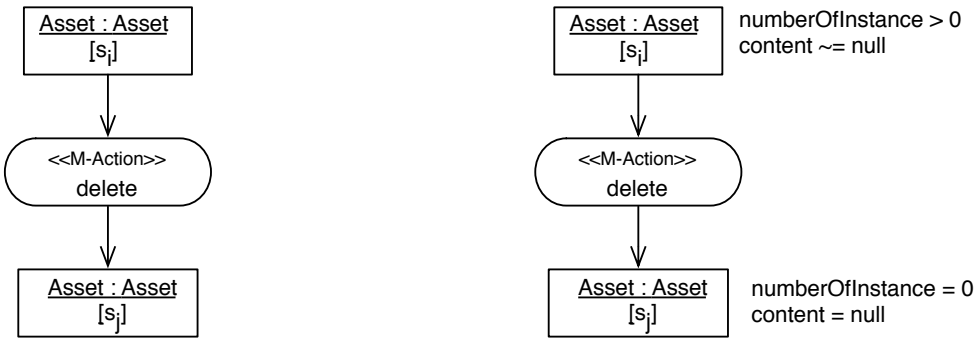
- *basic* threat-actions, are usually considered to be elementary if no decomposition will reveal any further information of interest.
- *composed* threat-actions, are obtained by composing actions to produce more complex behaviors.

Both basic and composed threat-actions can alter, e.g., the content of an asset, its state or both. Some actions are just aimed to obtain other malicious assets. However, they can trigger other more risky actions that can directly modify the asset, namely attack actions which do not change the current state of the asset.

Next, we discuss some examples of basic and composed threat-actions. Specifically, four *basic* threat-actions: *delete*, *read*, *write* (or *update*), and *create*, and two *composed* threat-actions: *replace* and *copy*. UML diagrams are used to give their intuitive meaning and the stereotype *"triggers"* in the diagrams underlines the fact that when a threat-action with no effect on an asset, but it can trigger one or more threat actions to take malicious effects. Notice that we always distinguish threat-actions from intended actions, discussed in the previous chapter, by using the *threat* or *threat-action* along with the action name (e.g., "delete" and "delete threat-action" refer two different intents.)

Examples of Basic Threat-actions. The purpose of a *delete* threat-action is to change the content of an asset, e.g., either by removing some the content or by destroying the asset itself in order to modify the asset-flow in the domain (Figure 4.3(a)). This corresponds to a partial or total effects on the asset when applicable. In case of a partial effect, the value of the asset is reduced by the value of the deleted quantity. In contrast, if the action has a destroying consequence, that is, a total effect on the asset, the state of the asset is changed to a special state called *"destroyed"* state. This means that the asset (instance) cannot be further used in other processes neither restored in some other way. Delete threat-action can be detectable or undetectable, but the destroying effect of the threat can results in to unrecoverable condition (i.e., detectable but not recoverable type of attack).

Figure 4.3(b) presents an example of delete threat in which *Delete* takes



(a) The *Delete* Threat-action. It changes the state of the asset from s_i to s_j.

(b) Instance of *Delete* threat action.

Figure 4.3: The *Delete* threat Action and an Example.

as input an asset with the number of instance greater than zero and *content* value different from *null*, and when the threat-action is successful, it resets the number of instance and the content to zero and *null* respectively. For example, deleting the content of a memory support (say, a USBKey), that contains electronic votes reduces the actual content and value of the USBKey, and consequently it can compromise the integrity of the election results.

The purpose of the *read* threat action is to obtain an intangible asset. An adversary observes an asset (e.g., what it contains or which value is assigned for the asset) and s/he reads the asset without altering its state. To execute this action, a domain asset is used. Namely, it takes a domain asset as input and returns the domain asset itself without altering the nominal asset-flow and a (subset of) malicious asset to be used later (see Figure 4.4). As a result, it can trigger other threat-actions, since the new information (i.e., malicious asset) can be used as input for other threat-actions.

An *update* threat-action has a goal to write a malicious asset to an existing asset. It takes a generic asset (could be malicious) and adds or

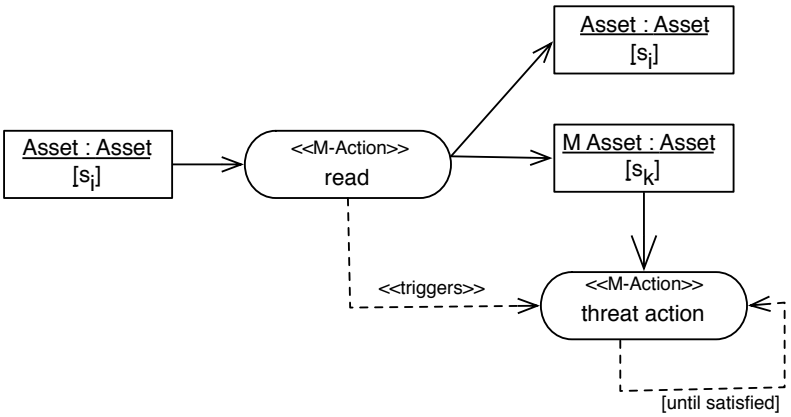


Figure 4.4: The *Read* Threat-action. The state of the input asset does not change, i.e., s_i.

introduces a malicious asset to the nominal asset-flow. In other words, it can change the content, value, location, or property of an asset by writing some malicious values onto these attributes. Notice that the input (malicious) asset by itself has no direct consequences on the asset-flow, but it is a mean to provide malicious information to update the generic asset. So, if an adversary observes a domain asset and fills an output asset of the same instance, then it updates that asset. This usually means that the state of the asset is changed, that is, the asset is altered. A similar threat-action with *update* is *write*.

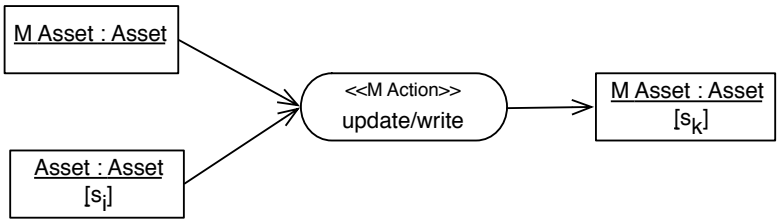


Figure 4.5: The *Update* Threat-asction.

Similarly, Figure 4.6 shows the *create* threat-action that builds a new malicious asset based on the information of a domain asset. Using this threat-action, an adversary can be capable of obtaining a new malicious

asset. An adversary may execute a *read* threat-action prior to executing *create* threat-action, since the read provides a malicious asset. It takes one or more assets from the domain and reintroduces the domain asset in the asset-flow, as well as it produces a new malicious asset that simulates the original behavior of the domain asset. Notice, however, that the *create* threat-action has no consequences on the domain asset-flow unless an adversary intercepts the nominal asset-flow with other threat-action using the resulting malicious asset. This is, in fact, what we call the *copy* threat-action (see below).

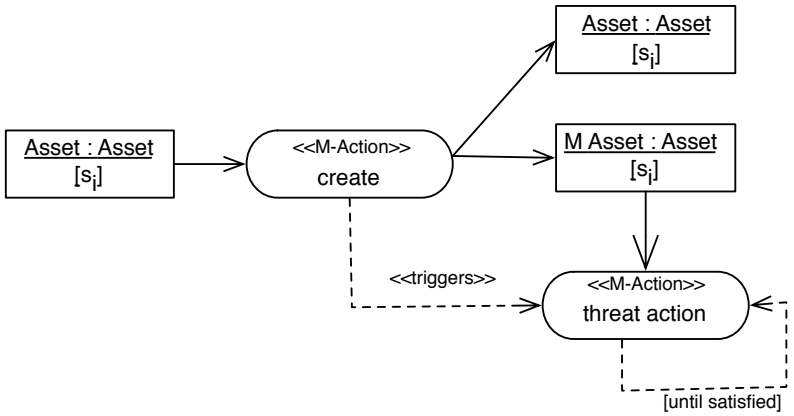


Figure 4.6: The *Create* Threat-action.

Composed threat-actions, examples. The goal of a *copy* threat-action is to produce a new copy of an asset by obtaining a malicious asset that can be used later. It is a composed action of *read* and *create*. Namely, after reading from an asset some (or all) its content or value, the asset is reintroduced into the nominal asset-flow and the read data is used to create a new malicious asset to be used later. To understand the difference between *copy* and *read* threat-actions, consider the following example. Assume a system protected by a password and smartcard device: To login with a password an adversary just needs a *read* threat-action and to access a smartcard protected system s/he has to *copy* the smartcard device. A

similar threat-action with *copy* is the *clone*.

Replace is another example of a composed threat-action. It substitutes an asset with other malicious asset. Replace takes as input two assets and introduces the second asset, that is, the malicious one is introduced. *Replace* action is a composed action of *delete* and *write*. That is, after deleting from an asset some (or all) of its content or value, the resulting malicious asset is again modified with wrong data and reintroduced in the asset-flow (Figure 4.8). The replace threat-action can be composed by *copy*, *delete*, and *write*.

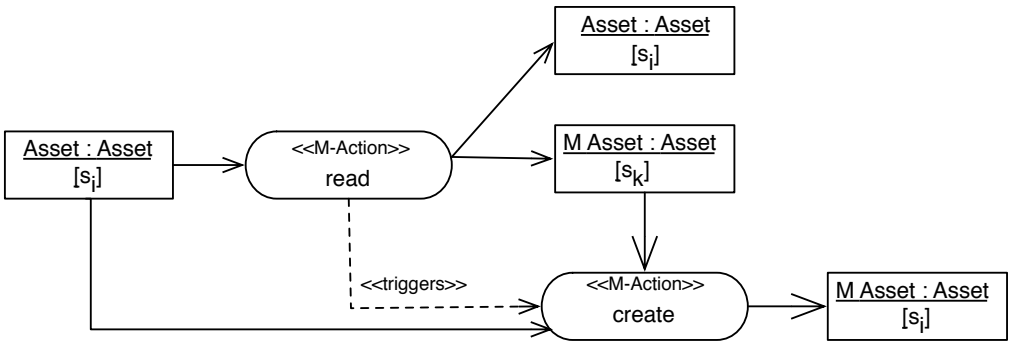


Figure 4.7: A simple copy threat action. It is composed by two threat actions: read and create.

In this way, it is possible to define more complicated threat-actions. To provide closer formal semantic, in fact, it is possible to use first order logic terms to denote (basic) threat-actions and provide recursive definition for composed threat-actions like (Koubarakis and Plexousakis 2000, Braynov and Jadiwala 2003). However, we are not interested in with this kind of representation. Rather, we will use the NuSMV input language to encode the threat-actions while extending the nominal assets-flow models.

Finally, to carry out an asset threat the adversary may need to execute one or more (more elementary) threat-actions or a sequence of asset-threats against other asset —e.g., adversary reads "password" and signs "data".

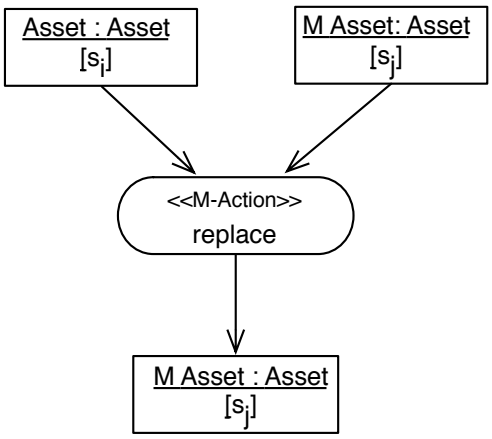


Figure 4.8: Replace threat action. It is composed by two threat actions: delete and write. Note that the composition detail is not shown in the diagram.

Therefore, we define an attack as a sequence of asset-threats that lead to an undesired state, that is, one or more assets are in an undesired asset-state.

4.3 A Methodology for Procedural Security

We developed a precise methodology to perform formal procedural security analysis (see also Figure 4.9). In the following, we discuss each element of the methodology in detail.

4.3.1 Formal Model of Asset-flows

The first activity in our security analysis of a procedure consists of providing models of the procedures in terms of assets-flow, which will be described using transition systems. Such models are meant to describe the assets to be analyzed, and are elaborated and transformed by procedures. We now briefly describe the terminology and the construct that allow us to arrive at executable specifications for the assets-flow.

The key elements in the model include assets, (workflow) activities,

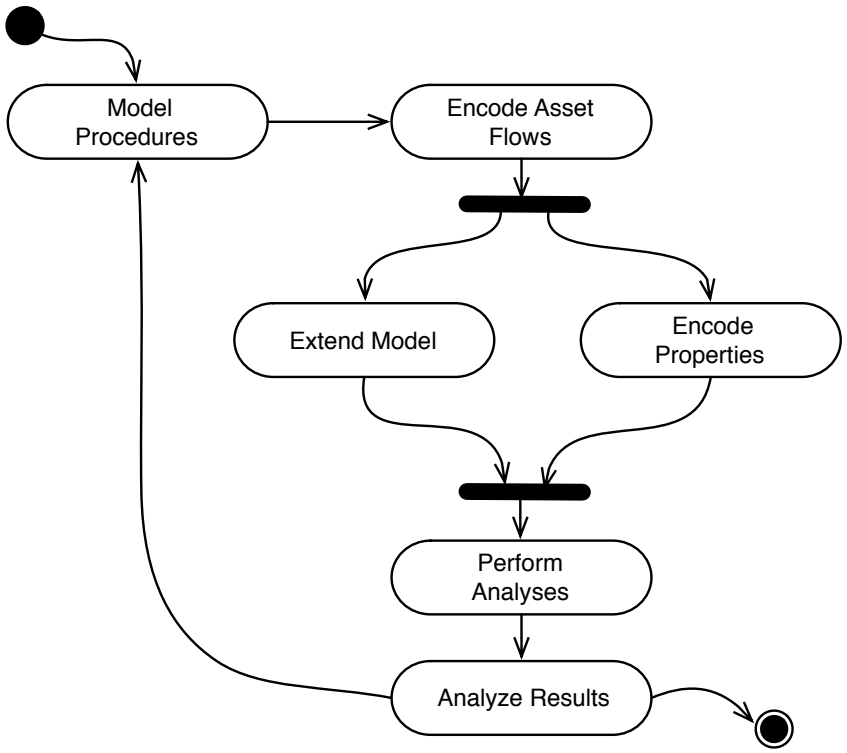


Figure 4.9: The process of formal procedural security.

and transition semantics, which influence the flows. Figure 4.10 shows high-level representation of the information and the behavioral —i.e., the lifecycle, models of assets. Unlike the classical UML activity diagrams, the perspective shown in the figure offers three views: workflow, assets class, and state machine diagram views. In the workflow diagram view, workflow activity sequences are defined. The state machine view describes the behavior of an asset in terms of a transition system in which transitions are enabled due to explicit execution of workflow activities. The activities in the workflow are transformation functions that influence the behaviors of the assets. A finite state transition diagram for each feature of an asset constitutes the global state machine for that asset.

For instance, Figure 4.11 shows a simple example of asset-flow model for asset instance $\underline{\mathbf{A}}$ with three states $[s_1]$, $[s_2]$, and $[s_3]$. The corresponding

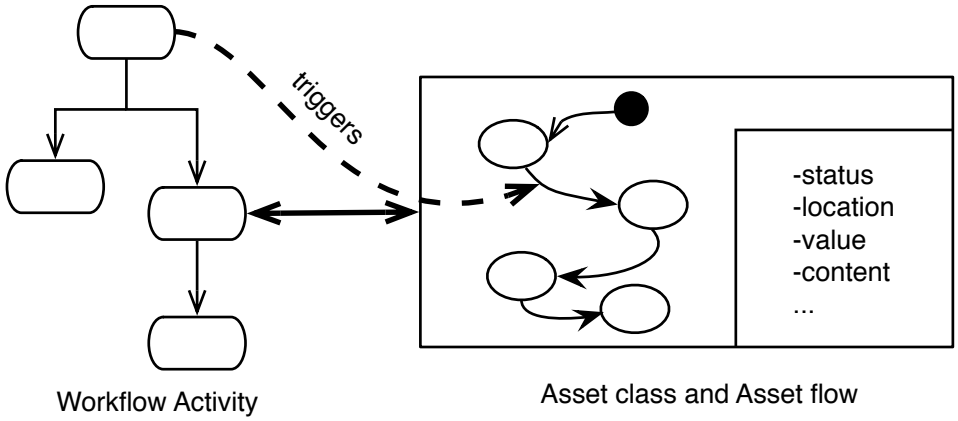


Figure 4.10: An asset-flow view of a business process model

finite state machine, therefore, will possibly have three sequential states each of which corresponds to **A**'s current features values.

4.3.1.1 Formalization of the Models.

The above information along with the conceptual model discussed in Section 4.2.1, are the basis for the formalization —namely, how we represent the assets structure and workflows to arrive at what we call executable assets-flow models. The formalization allows the model to be more amenable to formal analysis, since it removes the strategic flavor of the business process models and shifts the focus to dynamic aspects of the assets, and hence procedures under analysis. Our formalism borrows and extends the approaches proposed in (Gerede and Su 2007) and (Bhattacharya et al. 2007, Hull 2008). However, we differ on the interpretation of some concepts and on the analysis goals we wish to perform. In fact, our approach moves forward the cited works (in which the authors provide formal model for business artifacts and analyze them for better construction of business operations and processes) by complementing the their approach with security analysis.

To begin with, we assume the following notations and their definitions.

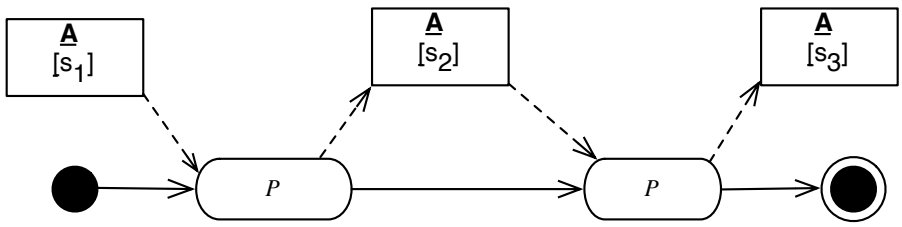


Figure 4.11: Example of a single instance Asset-flow model in three states.

- $\mathcal{T}_p$ be a set of primitive types, such as bounded integer and boolean;
- $\mathcal{C}$ be a set of asset classes (*names*);
- $\mathcal{A}$ be a set of attributes (*names*);
- $\mathcal{ID}_{\mathcal{C}}$ be a set of identifiers that describes the identifiers for each asset class $C \in \mathcal{C}$;
- $\mathcal{S}$ is a set of assets states, where each $s \in \mathcal{S}$ is a truth assignment over the variables values.

Note that all the above sets are finite, which is essential for the model checking process.

A *type* $\mathcal{T}$ is an element of the primitive types $\mathcal{T}_p$ and the class identifiers C; namely, $\mathcal{T} = \mathcal{T}_p \cup \mathcal{C}$ (we assume that $\mathcal{T}_p$ and $\mathcal{C}$ are disjoint).

Definition 4.3.1 *An asset class signature is a triple* $\langle C, A, \psi \rangle$ *where* $C \in \mathcal{C}$, $A \in \mathcal{A}$ *is a a set of attributes for the asset class* C, *and* $\psi : \mathcal{A} \rightarrow \mathcal{T}$ *is a total function that maps each attribute of an asset into its corresponding type.*

Definition 4.3.1 represents the asset class signature that specifies assets that are present in the domain. In the definition, besides the features of an asset discussed previously, the specification of the attributes can comprise of domain specific or assets specific attributes.

Notation 3 *We write Σ to denote the set of all possible assets classes that exist in the domain, including malicious assets that can be introduced by an adversary.*

Without loss of generality, we assume a fixed interpretation domain associated to each $\mathcal{T}$ type. That is the *domain* of each type $t \in \mathcal{T}$, denoted $\mathcal{D}^t$, is defined in the following way: if $t \in \mathcal{T}_p$ is a primitive type, then the domain $\mathcal{D}^t$ is some known set of *values* of type (e.g., integer or boolean); if $t \in \mathcal{C}$ an identifier type, then $\mathcal{D}^t$ defines existing instances of an asset class identifier for t (i.e., $\mathcal{D}^t = \mathcal{ID}_t$). We require all variables must have their corresponding *values* all along their life. For undefined location and unassigned content of an asset, we use an *undefined* and a *null* constant values respectively. The interpretation is that the location is not known and the content value is not either assigned yet or reset to contain null. For instance, the *content* of a memory support (such as USBKey) can be *null* prior to loading the election software by a responsible actor (e.g., by a technician or an election official).

Definition 4.3.2 *An asset instance is a triple $\langle ID_C, C, \phi \rangle$,* where $ID_C \in \mathcal{ID}$ is a class identifier and ϕ a partial function, given an instance of a class C, that assigns each variable $a \in A_C$ of type $t \in \mathcal{T}$ a value in D^t (i.e., $\phi(a) = D^t(\psi(a))$).

Notation 4 *We use $S_C \subseteq \mathcal{S}$ to denote all the possible states (i.e., both nominal and non-nominal states) of an asset class $C \in \mathcal{C}$.*

An asset can have multiple instances. We denote the set of asset instances by $\vec{\mathcal{O}}_{C,C\in\mathcal{C}}$ and $\vec{\mathcal{O}}$ for all instances over Σ. Duplicating the election software for each polling station, for example, creates as the number of polling stations instances where each of this instance can behave differently after the duplicate operation in place. As we noted previously, in

this work, we mainly focus on how a single asset instance $I \in \vec{\mathcal{O}}_C$ can evolve from some initial state through other states. The set of *variable-value* pairs for I defines the *state* of a given asset. The *state* of an asset is, therefore, the current situation called *"snapshot"* of an asset instance I and its value is the truth assignment over the variables. An asset is *initial*, if all the variables are in their initial state and ϕ is undefined for some attributes and *final*, if all the variables values do not change anymore.

Formal Model of Workflow specification. The initial values for the variables of an asset can be assigned at the time of the instance creation or otherwise assigned by an analyst. However, only due to the execution of a workflow activity over these variables can change the initial configuration of the asset. Roughly speaking, a workflow activity is described by input assets, preconditions, and effects of the activity over the assets (a similar interpretation can be found in (Bhattacharya et al. 2007, Gerede and Su 2007, Koubarakis and Plexousakis 2000)). The effect of a workflow activity is regarded as a change in state of the input assets. Not all assets change their states thought, since it is not always the case that an execution of a workflow activity enables state transition to all the input assets (e.g., reading the content of the election software does not change its state).

For each executable workflow activity, moreover, we specify which actors participate in the workflow with predefined privileges or responsibilities or both. These information —we call accessory information —not only allows to describe who does what during the execution of an activity, but, more importantly in the context of procedural security analysis, who manages what data and with what privileges. Such information are static, namely they are known before executing a workflow and are encoded in our model to describe a workflow scenario. We, therefore, use these information along with the activities to describe a workflow model as a deterministic finite state machine in which the states are constructed by a set of activities and

the transitions are described by the current state and a matching condition over the accessory information. Formally, we define the workflow model as follows.

Definition 4.3.3 ($\mathcal{W}$) *A workflow model is a quadruple* $\langle P, s_0, s_f, C, \Delta \rangle$ *where* P *is a set of activities or processes (names) as in the previous chapter;* $s_0, s_f \in P$ *are initial and final activities of the workflow respectively;* C *is guard expression over accessory information, and* $\Delta \subseteq P \times C \times P$ *is a transition relation between a current activity and its successor activities in which a transition is labelled with a condition over accessory information.*

It should be clear from the context that this definition is not meant to replace the definition of business process model discussed in the previous chapters (i.e., Definition 3.3.1) and/or elsewhere. Rather, the definition is meant to express the fact that there exist a set of activities within a particular workflow, that describes a procedure under analysis, in which by knowing the current state of the workflow, and if a condition is meet, it should be obvious to determine the next state of the workflow. We call an instance of a workflow model, a program counter *"pc"* that contains the value of the current state (i.e., the active activity) in the workflow. There is one program counter *"pc"* for each workflow model at run time. In actual business process or workflow specification, in fact, it is possible to have multiple activities that can run in synchronous or asynchronous mode. In this work we focus on sequential execution of a workflow.

Formal Model of Asset-flow specification. The state of an asset is specified by the assignments of values to variables —or simply valuations. This, in turn allows to describe the evolution of an asset. The evolution is expressed by the sequence of states through which an asset undergoes during the execution of a process. It makes sense to encode the state of each variable as a finite state machine, since the state of an asset is described by the

valuations over its variables. The workflow instances, along with some matching conditions, define transitions for modeling the lifecycle of the assets. Thus, an asset-flow can easily be modeled using a transition system, that facilitate the formal analysis. We now define our notion of assets-flow models more formally.

Definition 4.3.4 (Assets-flow model) *An asset-flow model (AFM) is a 5-tuple* $\langle AS, I, \mathcal{W}_\pi, C_\pi, \Delta_\pi \rangle$ *where*

- $AS \in \mathcal{S}$ *is a finite set of assets' (instances) states;*
- $I \subseteq AS$ *is an initial states of the assets;*
- $\mathcal{W}_\pi$ *is a set of workflow instances;*
- C_π *is a set of conditions constructed over the attributes representing the matching construct as a guard, that specify the condition must meet for the state to be changed, along with the current activity;*
- $\Delta_\pi \subseteq AS \times \mathcal{W}_\pi \times C_\pi \times AS$ *is a transition relation between a current state of an asset and its successor states in which a transition is labelled with an activity and a condition.*

A collection of individual AFM constitutes *assets-flow* models, and we represent it by $\mathcal{M}_1$. Therefore, $\mathcal{M}_1$ is regarded as the global configuration of the domain of interest, namely the procedures under analysis. The semantic of the global configuration $\mathcal{M}_1$ can be interpreted in the following way. Each $m \in \mathcal{M}_1$ is regarded as an abstract state machine, which has three major components: a workflow activity sequence (possibly maintained in a queue), a workflow activity dispatcher, and an activity processor. Workflow activities are added to the end of the activity queue. The activity dispatcher chooses, dequeues, and provides the next *"pc"* (i.e., an activity) to the activity processor. Each *"pc"* is then used as a *transformation*

function that can possibly change the state of an asset by modifying or changing one or more variables values of the asset. One state machine per feature variable encodes the lifecycle of that state variable (e.g., see Figure 4.14). A set of such state machines constitutes the global state machine for the corresponding asset instance. By defining a semantic for the state machines corresponding to each feature of an asset and linking it with $m \in \mathcal{M}_1$, therefore, we have implicitly defined how $\mathcal{M}_1$ behaves.

4.3.2 Model Extension

The formal model we presented earlier represent the *nominal* behavior of the assets ($\mathcal{M}_1$) —that is, the model describing the standard procedures, e.g., what is prescribed by the law. In order to analyze what are the possible attacks of a given (set of) procedures, we need to encode asset threats in the nominal model and generate the extended model $\mathcal{M}_2$. Structurally, in fact, there is no difference between $\mathcal{M}_1$ and the extended model $\mathcal{M}_2$. However, the main difference lies on the assets state *set* and on the transitions specification. This means that, the extended model possibly will have more states than the other due to the execution of threat-actions that can change the state of an asset into an undesired one.

On the transitions side, on the other hand, the definition did specify the fact that transitions are triggered only by nominal workflow activities. We need to incorporate in $\mathcal{M}_2$ the fact that an asset could be in any possible states and that such states can also be changed by the execution of malicious processes. However, it is pretty straightforward from the definition we gave and by extending the definition of the workflow model (e.g., Definition 4.3.3) to include all the malicious processes that an adversary might execute. Thus, in $\mathcal{M}_2$, assets are not only manipulated according to what should happen in the nominal case (i.e., according to the electoral laws), but can also be transformed by the execution of one or more assets

threat-actions.

Before providing the NuSMV encoding for $\mathcal{M}$, let us materialize those sequences of states that an asset can undergo. Not surprisingly, this corresponds to a *path*, that shows the evolution of an asset instance. Since our work mainly focuses in a single asset instance, thus, we speak about a path for a single asset instance I. Each *snapshot* s_I^i in the flow corresponds to a state described by the valuations of a given asset. And, s_I^{i+1} corresponds to the execution of a workflow activity in $\mathcal{W}$ (possibly with malicious processes) to the asset features. A path in $\mathcal{M}$ is a finite sequence π of states, $\pi = s_I^0, \ldots, s_I^n$, satisfying:

- Initiation: $s_I^0 = s_I^{init}$;
- Consecution: for every $i = 0, 1, \ldots, n$ the asset instance state s_I^{i+1} corresponds to the result of applying workflow activity and threat-actions to s_I^i, along with the satisfaction of some boolean constraints on the transition, if any.

4.3.3 Encoding the Assets-flow models in NuSMV

4.3.3.1 Overview of the NuSMV language

Before explaining the encoding of asset-flows modes in NuSMV, it is useful to provide an overview about NuSMV itself.

NuSMV (Cimatti et al. 2002) is a symbolic model checker originated from the reengineering, reimplementation, and extension of SMV (McMillan 1993). The tool has been designed to be a robust, well documented, open platform for model checking. It allows for the specification of synchronous and asynchronous systems and for the verification of safety and liveness properties expressed in temporal logics CTL (Computation Tree Logic) and LTL (Linear Temporal Logic), using both BDD-based and SAT-based model checker.

The NuSMV tool comes with a description language that is used to describe the (finite) fair transition system (specifically, that of Kripke transition systems) modeling the program under analysis. The description is broken down into `modules` that can be composed and reused. `Modules` describe initial values of variables and how they change in each step, as well as they can have parameters which can be used to establish identities or connections among other modules.

The type of a variable within a module can be boolean, enumerated, finite range of integers given by <number> ... <number>, or finite arrays. The keyword `init` is used to describe the initial value of a state variable and unspecified variables can take any value in their type as the initial value. The `next` construct describes how the value of the variable changes in one step. If the `next` value is unspecified, then the variable takes any value in its type at the next step. The `case` statement assigns the value associated with the first case condition that is true currently, or otherwise 1 (one) as the default case.

The language also allows to define a C-like "macros" using the `DEFINE` construct. The variables defined using the `DEFINE` construct do not have any effect on the state space, since they are not a real variables. Finally, the language and the machinery have demonstrated to be adequate for several industrial sized systems. In our case, the choice of the tool has been driven by practical considerations (it is a tool for which we have extensive know-how), stability (it is used by a wide community), and availability (open source[3]).

4.3.3.2 Encoding the Assets-flow models

Assets-flow models $\mathcal{M}$ can become executable specification to allow formal analysis through verification tools on their evolution, including their ma-

[3] `http://nusmv.fbk.eu/`

licious evolution due to threat-actions. Our aim here is to represent the model $\mathcal{M}$ into executable specification using NuSMV input language. We mentioned that the NuSMV semantic is based on a state-based formalism in which the behavior is defined by Kripke transition systems. However, the above definition for $\mathcal{M}$ is an action-based formalism in which the behavior is defined by (a sort of) labelled transition systems. Thus, we need to rearrange the previous definition to align with the semantic of Kripke structure so that the encoding of NuSMV specifications can be tackled.

Definition 4.3.5 (AFM, $\mathcal{M}_K$) *Let APs are set of atomic propositions ranged over some boolean expressions on the valuations of the variables. An asset flow model (AFM) is a Kripke structure over a set of atomic propositions AP defined by a quadruple $\langle AS_K, I_K, \Delta_K, \mathcal{L}_K \rangle$ where*

- *AS_K is a finite set of assets (instances) states;*
- *$I_K \subseteq AS_K$ is set of initial states;*
- *$\Delta_K \subseteq AS_K \times AS_K$ is a transition relation between a current state of an asset and its successor states;*
- *$\mathcal{L}_K : AS_K \rightarrow 2^{AP}$ is the labeling function which returns the set of atomic propositions which hold in a state.*

Notice that the main variation between the previous definition and the above definition is on the encoding of the transition relation. In the former case, the transition is explicitly encoded by the workflow activity and a condition. In contrast, in the latter case, it is specified either into the source state of the transition as a boolean expression, or into the target state of the transition as assignment statement(s). We should be clear that this kind of rearrangement is not new (e.g., a similar work can be found (Lam and Padget 2004)).

Therefore, the encoding of $\mathcal{M}$ in the NuSMV input language can be treated as a problem of defining a mapping between the two structures, i.e., between the structure specifying the model $\mathcal{M}$ and the Kripke structure.

In general, the translation is performed as follows:

1. we define a module that works as the program counter, that is, it encodes the sequence of activities defined by the workflow. are executed is the one defined by the workflow. The transition from one activity state to the next is determined by the current state of the activity and some accessory information such as role(s) (which encodes the relationship between the workflow activity and actors who are assigned to perform the execution of the workflow).

2. a module is defined for each asset that it is specified in the assets-flow models (or specified in the process diagram). Each feature of the asset is defined as a state variable within the module. The transition from one asset state to the next is determined by the program counter (which represents the execution of an action of the workflow) and some boolean expressions over the current state of the asset;

3. we define some boolean appendage variables to capture malicious flows of assets and the execution of threat actions to form the extended model. That is to say, these variables help for the encoding of both malicious assets and threat actions to derive the extended NuSMV model. Additionally, we introduce a state variable in workflow module to capture malicious processes.

The above general strategies are organized into a number of rules. More specifically, the following encoding rules can be used to map $\mathcal{M}$ into the NuSMV counterpart.

Rule 1 *The workflow model is encoded in NuSMV as a special module, and each workflow activity $p_i \in P$ for $i = 1, \ldots, n$ representing the domain activities (i.e., processes) in $\mathcal{W}$ are encoded in the NuSMV input language as a scalar variable program counter (*pc*) in which p_i are its symbolic values.*

```
MODULE Workflow (...)
    VAR
        pc : {p_1, p_2, ..., p_n};
```

Rule 1 defines the special module that works as the program counter. In particular, it specifies the workflow model ($\mathcal{W}$) and the declaration of state variable pc under the module, where all domain activities in the workflow are the scalar values of pc. The pc ensures that the order in which activities are executed is the one defined by the workflows.

In order to determine the state transition of the program counter, we introduce some predicates (see Table 4.1) from the business process model presented in Chapter 3. Notice that these information can easily be inferred by looking at the process diagrams. They are mainly associated with the accessory information, such as actor-role and actor-activity (i.e., R_{AP}) assignments. The table also shows the corresponding state variables in NuSMV input language.

Predicate	**Meaning**	**NuSMV variable**
AssignR(a,r)	assignment of actor $a \in Actor$ to role $r \in Role$	assign_a_r
AssignA(a,p)	assignment of actor $a \in Actor$ to an activity $p \in P$	assign_a_p
r_Active_for_a	role $r \in Role$ is active for actor $a \in A$	activefor_a_r
ExecA(a,p)	actor $a \in A$ executes an activity $p \in P$	exec_a_p

Table 4.1: Accessary information as predicates.

Rule 2 *The accessory information are encoded in the NuSMV input language within the Workflow module in the following way (see also Table 4.1):*

- *For each actor-role assignment, we introduce a variable assign_a_r. assign_a_r is true iff the predicate AssignR(a,r) is true for an actor $a \in Actor$ and a role $r \in Role$, where Actor and Role are as Chapter 3;*
- *For each actor-process assignment, we introduce a variable assign_a_p. assign_a_p is true iff the predicate AssignA(a,r) is true for an actor $a \in Actor$ and an activity $p \in P$;*
- *For each role activation r_Active_for_a, we define a state variable Activefor_a_r;*
- *Similarly, we define a variable Exec_a_p for every actor performing an activity, i.e., iff ExecA(a,p) is true.*

Rule 2 defines accessory information for the transition relation of `pc` state variable. Notice that activities can only be executed if the activity instance in question is assigned to an actor —i.e., ExecA(a,p) ⇒ AssignA(a,p). Moreover, a group of actors can perform the same activity, as discussed in the previous chapter.

Rule 3 *For each asset instance in $\vec{\mathcal{O}}$, a NuSMV module is defined:*

```
MODULE ASSET_NAME  (...)
```

Rule 4 *An asset with no content in $\mathcal{M}_1$ is mapped to a symbolic value* "null" *in NuSMV. Similarly, an asset whose current location is not known or unspecified in $\mathcal{M}_1$ is mapped to a symbolic value* "unspecified" *in NuSMV.*

Rule 5 *The location, representing all the possible places of an asset can be, is encoded in the NuSMV input language as scalar variables* `loc` *in which* loc_i *for* $i = 1, \ldots, n$ *and "*$undefined$*" are its symbolic values. The content, representing all the contents of an asset at a particular point of time, is encoded in the NuSMV input language as* `content` *in which* $content_i$ *for* $i = 1, \ldots, n$ *and "null" are its symbolic values. The value, representing all security risk values for an asset, is encoded in NuSMV input language as* `value` *in which* $noValue$*, low, high and critical are its symbolic values. Finally, each domain specific* `property` *of an asset in an asset-flow model is encoded as a boolean value in NuSMV.*

```
MODULE ASSET_NAME  (...)
  VAR
     loc :  {loc_1, loc_2, ..., loc_n, unknown};
     content :  {content_1,content_2, ..., content_n, null};
     value :  {noValue, low, high, critical};
     property_1, property_2, ... : boolean;
  ...
```

Rule 3 states that a module is defined for each asset (instance) in $\mathcal{M}_1$. In Rule 5, whereas each feature of the asset is defined as a state variable within the asset module specification. An *unknown* `location` and a *null* `value` are both encoded by symbolic values as defined by Rule 4.

Rule 6 *The transition specification for each state variable is encoded by the current value of the program counter and some boolean expressions over the current state of the asset. The below code shows a template for a transition specification for each state variable.*

```
MODULE ASSET_NAME (pc, ...)
  [...]
```

```
/*template for content transition encoding.*/
  next(content) :=
    case
        pc.pc = activity &  bool_expr: content_j;
       [...]
      1 :  content
   esac;
[...]
```

The above rule (i.e., Rule 6) encodes the transition specifications. The transition from one asset state to the next is determined by the current value of the `pc` and some condition over the current state of the asset instance. That is, the current state of the `pc` —which is passed as a parameter to each asset module— along with boolean expression, *bool_expr*, is encoded as a boolean expression like `pc.pc` = *activity* & *bool_expr* from $\mathcal{W}_\pi \times C_\pi$.

Model Extension. Since all the above rules are related to the encoding of $\mathcal{M}_1$, we need to provide additional rule for encoding $\mathcal{M}_2$. The model extension corresponds to proving an extension in the NuSMV model with one or more applicable attack-actions. That is, a specification of how the assets can be in undesired states. This can be done by associating threat-actions with variables defined inside the module per asset instance. Moreover, the `Workflow` module should also need to be extended in order to include the malicious process executions.

Note that attacks depend on what threat-actions are carried out, the effectiveness of the analysis depends upon the injection strategy that is chosen. It turns out that the best injection strategy consists of injecting all possible threat-actions at all possible steps of the nominal procedures and let the model checker to find the possible combination of the sequences that lead to undesired state for an asset flow. Therefore, the problem of

encoding of asset threats corresponds to extending the nominal assets-flow specification with threat actions.

In particular, the model extension can be done by using the following strategies:

- by defining a scalar state variable to encode all the possible malicious process within the Workflow module. Therefore, the program counter not only can have values from nominal workflow activities but also from possible set of malicious workflow activities;
- by defining a transition specification for each activation of a threat-action on asset instance under the corresponding asset module in NuSMV, where the malicious activity (i.e., the current value of the `pc`) is in place for enabling the transition;
- by defining boolean variable to monitor the execution of the corresponding threat-action. This variable will be true iff when the corresponding threat-action takes place;
- by introducing a scalar value "garbage" for the content state variable related to the introduction of malicious asset and a boolean variable. This variable will be true iff a predicate associated with the action (e.g., MAsset(t)) is true by a threat-action, say t.

The above strategies facilitate the task of model extension, by adding a number of boolean appendage variables that are needed to capture the malicious asset flows and the execution of threat actions to form the extended model specification in NuSMV input language. That is to say, these variables help for the encoding of malicious processes, malicious assets, and threat-actions to derive the extended NuSMV model. In this way, therefore, the model extension is performed for each applicable threat-action against the normal flow of assets. Notice that the model extension can be

done in two ways with different abstraction levels, namely either at higher level (i.e., at UML diagrams level) or at lower level (i.e., at the NuSMV specification level). The list of activities executed to carry out, e.g., an attack, we can derive the list of actors involved, simply by looking at the UML activity diagrams.

4.3.4 Property Capturing and Model Checking

During this phase, an analyst defines (procedural) security properties that will be used at a later stage to assess the behavior of the procedure under analysis. More specifically, assets-flow model definition, asset attacks definition, and model extension are just a part of the verification and security analysis process. Formal verification is carried out by defining properties in the form of temporal specifications.

We use LTL and CTL to encode security properties. LTL allows to specify properties related to each possible state of a system along a path —namely, to reason on the computational path scenarios of an asset (e.g., *"what can happen as asset travels along different locations"*). By contrast, CTL is used to specify properties related to how the state of a system can evolve overtime along all the possible computational paths —namely, to reason about the existence of specific states (e.g., *"is there any particular state in which an asset can be altered in an undesired way"*).

In fact, the types of properties to specify depend on the goals of analysis we wish to perform on the models. We are interested, in particular, in the following classes of properties:

1. The **Actor-Play-Role**, namely the roles actors have in the execution of the attack and the privileges they get on assets.

2. **Undetected attacks**, namely sequence of actions that succeed in altering one or more assets and for which the procedures provide no

check to highlight the alteration.

3. **Denial of services**, namely attacks which are meant to alter one or more assets in such a way that procedures have to be stopped. In the optimistic case, a denial of service in an election represents a cost and a "nuisance" for the community (as, e.g., results are delayed; the administration needs to re-run the election). In the pessimistic case, e.g. repeated attacks, it may represent a serious threat to democracy.

4. **Reachability analysis**, namely the sequence of actions leading to the violation of a security goal, with particular respect to the execution of asset-threats.

Once all the properties of interest are specified with respect to the analysis goals described above, the next activities are formal verification and analysis of the results. That is to say that as long as a nominal model ($\mathcal{M}_1$) or extended model ($\mathcal{M}_2$) is available, it is possible to verify its behavior with respect to the desired CTL/LTL properties.

The model under analysis is checked (i.e., model checking) against the security properties using NuSMV. And, the results of the analysis can be used for further discussions. In the case of a system property, the model checking engine can test validity of the property, and generate a counterexample in case the system property is proved to be false. For instance, if we consider a property that is required to hold for every possible path of the asset-flow (CTL property), the model checking engine will generate a counterexample showing one particular path along which the property has failed. In standard situations, the counterexample will contain the execution of one (or more) asset threat.

A counterexample in which no asset threats are executed would show an inherent weakness in the *nominal* workflows or otherwise a result of poor specification. The counterexamples of security properties encode sequences

of actions that, if executed, pose a threat to security of one or more assets. Furthermore, before calling the verification engine, it is possible to perform constrained or random simulation and several kinds of formal verification analyses using the facilities provided by the NuSMV engine.

4.4 A Case Study

We now demonstrate our approach in a real scenario in which part of a complex procedure that is followed to run election in Italy. That is, an excerpt procedure followed during the e-voting project trials using the ProVotE e-voting machine.

Provide models of the procedures under evaluation. Figure 4.12) shows a subset of UML diagrams that is used to model (an excerpt of) the procedure that is followed during project trials for delivering the voting software to the polling stations, as used in the ProVotE case.

The diagram shows how, before the election, the Electoral Office encrypts the e-voting software and creates a memory support which contains the final software release. The responsible person at the Electoral Office prepares an envelope with the PIN code (that it is used to activate the voting functions) and the memory support. We classify Electoral Office's into different level of technicians, for example, technician for generating encryption key and another technician for preparing the envelope. A messenger (e.g., a police officer) takes the envelope and delivers it to the polling station, where the polling officers, once verified that the enveloped is sealed, opens it, insert the memory support in the voting machine, insert the PIN and start the voting operations.

We then injected threats into the model of the procedures that it forms the extend model. Figure 4.13 depicts the extended model resulting from the injection of some *delete* and *replace* threat-actions in the example of

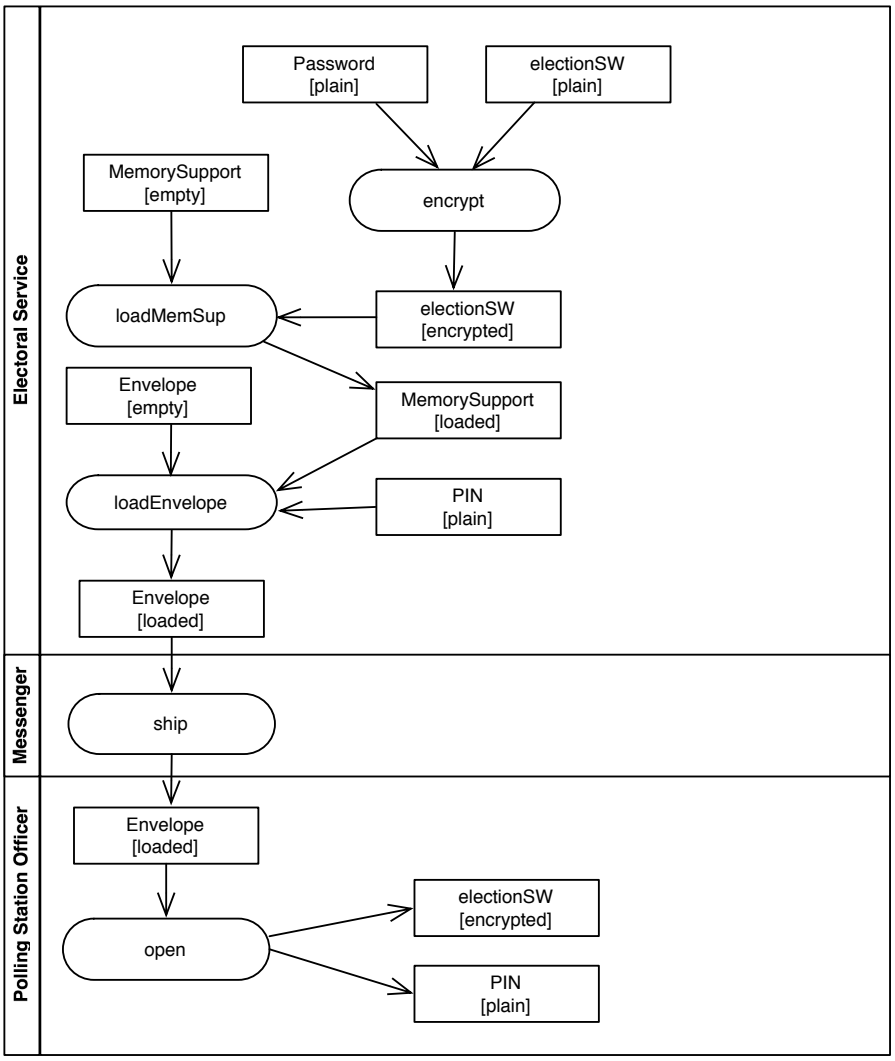


Figure 4.12: An example of asset flows.

Figure 4.12. Note that the semantics of delete and replace actions may slightly vary when applying them to different asset kinds. In the extended model, we marked threat actions with the `threat-action` stereotype. As noted before, the model extension (or threat injection) step is not necessarily be done after the process modeling step; it can also be done after encoding the asset flows.

NuSMV Encoding. We model the asset-flows into executable specification using the NuSMV input language using the translation rules described

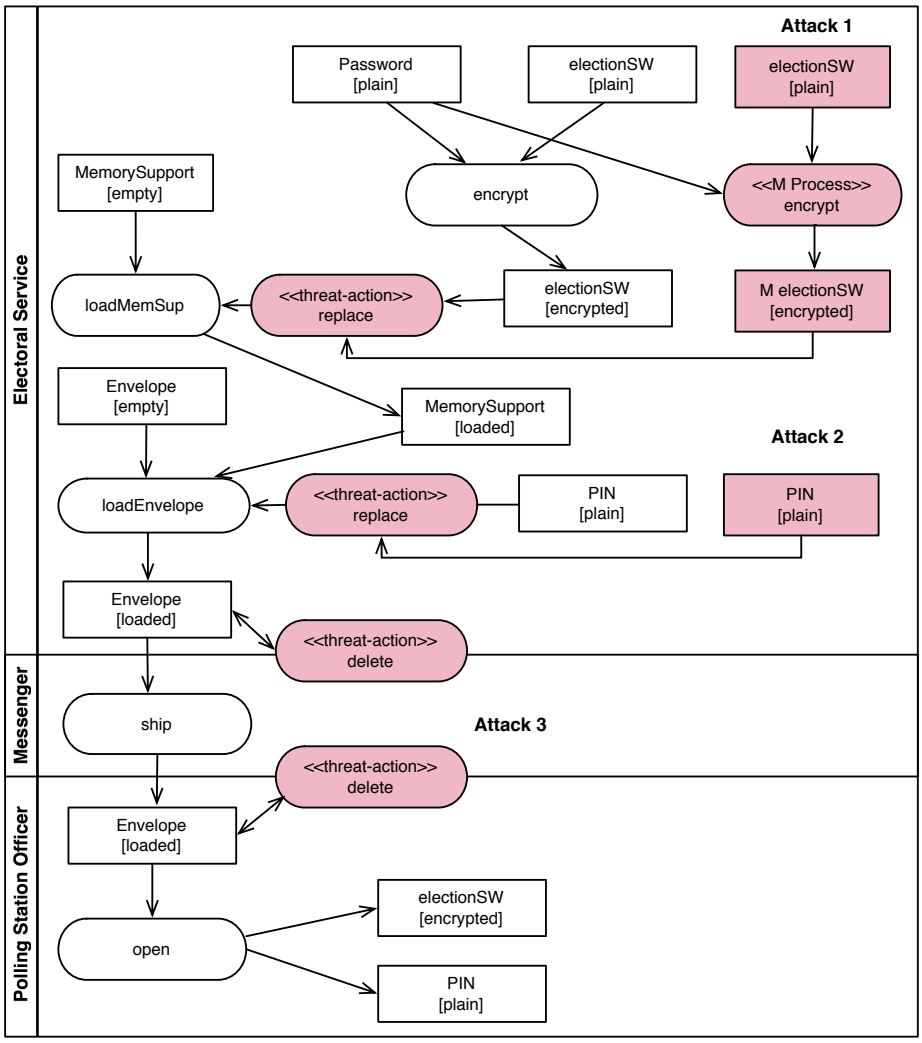


Figure 4.13: An example of extended model for Figure 4.12, where the introduction of the attacks are colored. It shows delete and replace threat-actions change the flow of the procedure under evaluation.

previously. We declared four modules corresponding to each assets in the diagram (Figure 4.12): *Password*, *electionSW*, *MemorySupport*, and *PIN*. The following snippet of code defines the asset type `electionSW` and some of its features, named `status` (that is, the states in which the `electionSW` can be), `value` (the relative weight of an asset assigned based on the criticality of the asset, basically the analyst decide and assign this value), and `content` (that is, the qualitative value of the `electionSW` can be), etc.

```
MODULE electionSW ( ... )
 VAR
   status  : {Plain, Encrypted};
   value   : {NoValue, Low, High, Critical};
   content : {PlainSW, EncryptedSW, SignedSW, EncryptedEnvSW,
              null},
   loc : {ElectoralOffice, PollingPlace, PoliceOffice,
          Messenger, Undefined},
```

Similarly, other modules with their corresponding feature variables are declared (e.g., `MODULE Password(...)`, `MODULE MemorySupport (...)`).

There are five (domain) activities in the nominal assets-flow diagram (see Figure 4.12), namely it contains activities `encrypt`, `loadMemSupport`, `loadEnvelope`, `shipEnvelope`, and `openEnvelope`. We declare a module called `Workflow` and specify state variable which have these five scalar values. In other words, the `pc` state variable within the `Workflow` module has five possible scalar values.

```
MODULE Workflow ( ... )
 VAR
   pc : {encrypt, loadMemSupport, loadEnvelope, shipEnvelope,
         openEnvelope};
   execute_actor2_encrypt : boolean;
   execute_actor1_openEnvelope : boolean;
  [...]
```

"Accessory" information (not strictly necessary to execute the workflows), such as the actors responsible for each activity, is encoded in the model through `DEFINE`s in the main module, such as in the following snippet:

```
DEFINE
   ElectoralServiceActive_for_actor3 :=
```

```
            pc  = loadMemSup || pc = loadEnvelope || [...]
POfficeActive_for_actor1 := pc  = openEnvelope || [...]
```

Evolution of assets' properties are encoded using state machines, which are encoded in NuSMV with the `next` construct (which specifies the value of a variable at step $n + 1$, given the value at step n). Notice that asset flows are defined both in terms of the program counter (e.g., the current step of the workflow) and the value of the asset features. Figure 4.14 shows a model of feature `content` variable of `electionSW` where its state changes according to the program counters and according to the current values of some state variables; its corresponding snippet NuSMV code is also shown.

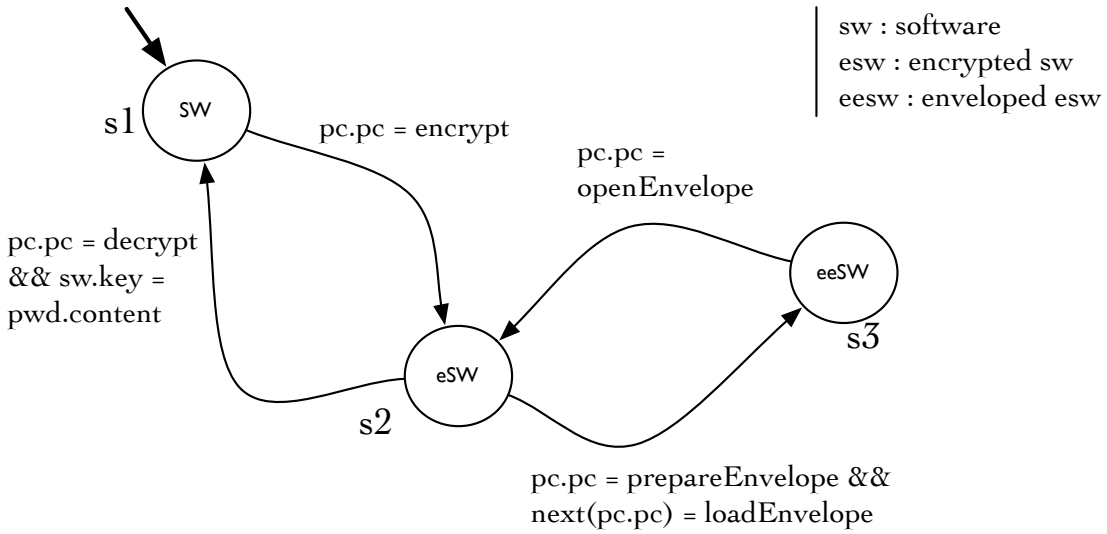


Figure 4.14: A simple example of state transition model for `content` feature of `electionSW`.

Note that in the code shown below, we have left some detail specification for the matter of presentation purpose.

```
[...]
 next(sw.content) := case
```

```
    (pc.pc = encrypt &&  content =  PlainSW && loc = ElectoralOfffice)
    || (content = EncryptedEnvSW && pc.pc =  openEnvelope
       && POfficersActive && loc = PollingPlace)
    : EncryptedSW;
    (pc.pc = loadEnvelope && (TechnicianTwoActive
    || ElectoralServiceActive) && content = EncryptedSW)
    : EncryptedEnvSW;
    pc.pc = decrypt && POfficersActive && (status = Encrypted ||
    content = EncryptedSW) && !fakeKey
    : plainSW;
[...]
```

Model Extension. Next, we show the extension of the model according to the diagram depicted in Figure 4.13. Threat injection (model extension) corresponds to augmenting the state machine of the asset flow with new transitions corresponding to the execution of threat-actions. Figure 4.15, for instance, shows an asset flow with some threat-actions that may alter a feature of an asset (e.g., `content`), in some undesired way.

The triggering of a threat-action is monitored through boolean variables that are set to true when the action takes place, as illustrated by the following pieces of code. We first declare one boolean variable per threat:

```
can_mesw, can_meesw: boolean;
can_garbageSW: boolean;
can_mPPin, can_mePPin: boolean
```

The above variables are initially set to false. When a variable is set to true (either because constrained to do so by the model or, more often, at random), a transition in the state machine encoding the asset flow is triggered and the value of the asset flow changed according to the threat-action (rather than to the nominal flow), as illustrated by the following piece of code:

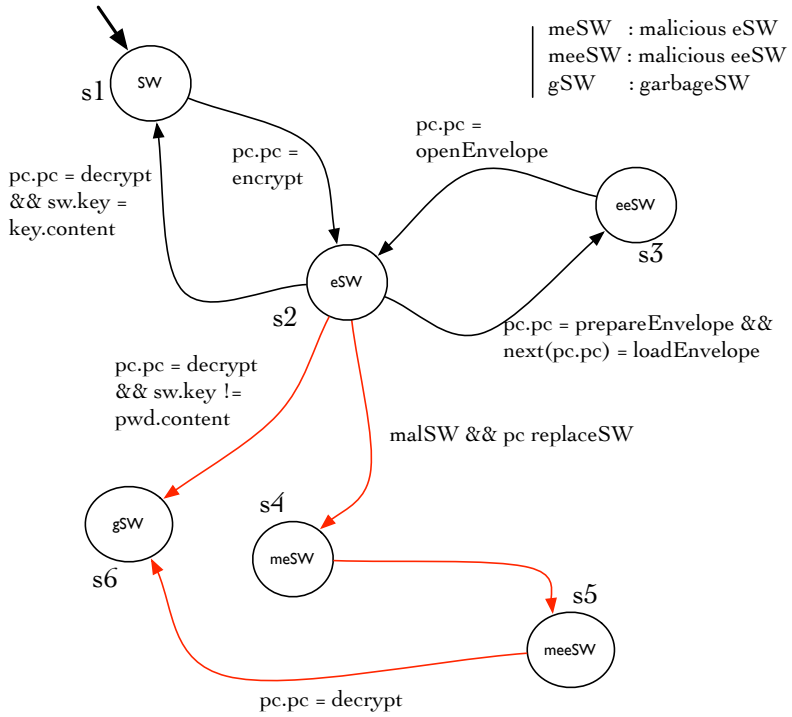


Figure 4.15: An extension of Figure 4.14 due to the injection of threat-actions.

```
    next(can_mesw) := case
     (malSW && pc.replaceSW && next (pc.pc) = loadMemSup
      && loc = ElectoralOffice)
     || (can_meesw && pc.pc =  openEnvelope && loc = PollingPlace)
     :1;
    1: can_mesw;
esac;
```

Beyond the above boolean variables, we define control flow variables which govern the execution flow and correspond to the introduction of malicious assets. For example, we have introduced a boolean variable `malSW` indicating that we deal with the introduction of malicious `electionSW`. Due to the fact that we do not want to restrict this variable in advance and that on the other hand the variable should be constant during the whole execution, we use the following trick of specifying `next(malSW) := malSW` without initialization. This means that we can choose the value of `malSW`

for the introduction of malicious `electionSW` at random, but once chosen, the value does not change anymore.

Property Capturing and Model Checking. After encoding the relevant information of asset-flows in NuSMV, the next activity consists of capturing and specifying the security properties using LTL/CTL. The security properties we wish to specify are with respect to the analysis goals we discussed earlier. For instance, in the example we have shown (in the extended model) it is possible to implement at least three different attacks.

- The first one consists of replacing the software which is sent to the polling stations. By reading the `password` with which the `electionSW` is encrypted and substituting a modified version of the software in the `MemorySupport`, it is possible for a malicious actor eventually to deliver a modified copy of the software to the polling station.
- The second one consists of replacing the `PIN`. A malicious actor with access to the `PIN` code may substitute the `PIN` which is loaded in the `envelope`. Thus, a wrong `PIN` is delivered to the polling station which eventually causing a denial of service —namely, the voting functions cannot be activated by the polling officers.
- The third attack consists in deleting (or destroying) the `envelope` during transportation, possibly causing another denial of service.

We show two examples of properties that allow us to highlight such attacks.

i) Verifying a property about the delivery of election software. In this example, we want to check a generic property about the delivery of software to the polling station. This allows us to derive the possible sequence of threat-actions following the counterexample generated by the NuSMV tool, if the property is proved to be false. We are interested in checking that: *"It is never the case that poll officers receive an altered*

election software". In other words, this property says that election software always remains "useful". The property is specified in CTL as:

```
AG ! (sw.content = garbageSW && sw.location = pollStation
     && PollOfficersActive )
```

When checking the above formula in NuSMV, it proves to be false, which is indicated by the counterexample shown in Table 4.2. Namely, it is possible to deliver a wrong software to poll station as illustrated by the counterexample, that is, sequence of attack-actions. In the real scenario, the flow should be: "After the envelope is opened, immediately afterwards the content of the software must be encrypted and the software must not be modified". The key point here is that, at time t_2 the software is encrypted, at the same time a malicious encrypted software is introduced (`malSW`) followed by replace attack action (`replaceSW`) before loading the nominal encrypted election software into the memory support. Then, at time t_3 `can_mesw` is true —that means, a maliciously encrypted software is introduced and loaded into memory support after being replaced by correctly encrypted software.

	t_0	t_1	t_2	t_3	t_4	...	t_9	t_{10}	...
pc.pc	—	—	encrypt	loadMem	preEnv	...	openEnv	decrypt	
sw.conten	sw				garbage	...			
malSW	$\bot$		$\top$	$\bot$		...			
replaceSW	$\bot$		$\top$	$\bot$		...			
sw.can_mesw	$\bot$			$\top$		...			
sw.can_garbage	$\bot$				$\top$	...			
sw.is_sw	$\top$				$\bot$	...			

Table 4.2: Example 1: A counterexample showing the alternation of election software at poll station.

It is easy to derive what are the possible scenarios in which an adversary

can invalidate the election software, given the above sequence of actions in place. From the property failure above (at time t_4), we can say that after the replace attack action takes place, the content of `electionSW` becomes a `garbage` right after loading it into the memory support, and at time t_9 and onward the software is indeed `garbage` (that is, it is useless). In this way, the counterexample is used to highlight sequence of actions in which the software has been altered during the delivery process.

ii) Verifying a property about the Denial of Service attack. In this example, we are interested in checking a denial of service attack that could happen in a poll station. Note that our aim here is also deriving the possible sequence of actions that an adversary can take to cause this attack. The property of interest is that: *"It is never the case that poll officers get denial of service due to PIN code"*. Notice that the PINs are used by poll officers to activate the voting machines to activate voting functions. This property is expressed in CTL formula as:

```
AG ! (PIN.can_garbage &&  PIN.location = pollStation)
```

We give the above property to NuSMV to check that the property holds. However, the tool generates the counterexample depicted in Table 4.3. Upon analyzing the generated counterexample, the PIN is replaced following the insertion of a wrong PIN into the asset flow, which, in turns, causes denial of service attack —namely, the poll officers unable to activate the voting machine due to wrong PIN. The scenario is as follows, at time t_4 malicious PIN code is introduced (`malSW`) while the electoral office is preparing the envelope, at the same time an adversary implements replace attack (`replacePiN`) before loading the envelope with memory support and the PIN code, eventually the PIN code is indeed useless causing the denial of service attack

	t_0	...	t_4	t_5	...	t_7	...	t_{10} ...
pc.pc		...	preEnv	loadEnv	...	openEnv	...	
malPiN	$\bot$	...	$\top$	$\bot$	...		...	
replacePiN	$\bot$	...	$\top$	$\bot$	...		...	
pinReady	$\bot$	...		$\top$	...		...	
can_mPPiN	$\bot$	...			...	$\top$	...	
pin.can_garbage	$\bot$	...			...		...	$\top$

Table 4.3: Example 2: Denial of service attack counterexample.

The examples, although trivial, show how —by reasoning on the extended model— it is possible to explicitly represent the attacks that can be carried out, determine what assets are needed by the attackers and when, and who can carry the attacks. Similarly to what happens in model checking, we do not provide any quantitative information about the likelihood of the attacks. However, even in this simple case, we believe that the output of the attacks can provide experts the information and the requirements to enhance the current procedures, to eliminate certain attacks or, at least, to make them more difficult to implement.

4.5 Summary

In this chapter we have demonstrated the importance of procedural security to tackle the security risks associated with the e-voting systems and eventually strengthen the level of security. Since asset mobility, state, evolution, and the context in which asset instances are used in e-voting are an inherent challenge, we have developed assets-centered methodology for procedural security analysis. The methodology can be used to analyze and evaluate the impact of threats, and consequently to come out with a set of (security) procedural requirements that guarantee the desired level of

protection. We presented the approach by introducing the guidelines we follow for modeling, and encoding the electoral procedures and hinted its usage through example.

Interestingly, the methodology we developed can allow for a systematic analysis of (procedural) security based on explicit reasoning on asset flows. That is, by building a model to describe the nominal procedures under analysis and injecting possible threat-actions by assuming that any combination of threats can be possible in all steps into the model. We also outlined encoding strategies using NuSMV input language —that it is amenable for formal analysis allowing to reason on different properties of the procedure on the extended model such as, the "actor-play-role" principle and "reachability" of (un)desired state of an asset.

The outputs of the model analysis using model checking technique helped us in understanding threat compositions on the asset-flow, and consequently to the procedures that take place when an adversary builds threat from the micro-level threat actions. Certain patterns can be combined to understand the level of coordination in order to understand undetectable attacks. For instance, an actor with privilege to read plain password and another actor (or set of actors) with privilege to access software together could result in replace composed attack.

Along with the approach discussed in Chapter 3, the procedural security analysis methodology has been applied within the ProVotE project. Within the project we have analyzed the existing laws and procedures, defined the procedures and developed the (e-voting) system that has been used in local elections —mainly for experimental purpose. The results of the analysis has led to enrich and improve the business process models that define the *to-be* system description for ProVotE e-voting system, and consequently for the development of the system itself. Such results are foundations to familiarize actors with the possible procedural threats and attacks that

can happen in elections. Additionally, this kind of analysis and reasoning has the potential to serve as a trust building measure in the new e-voting processes. And yet, the methodology and the analysis are the basis for system level reverse synthesis on currently deployed e-voting systems which will be discussed in Chapter 5. Therefore, procedural security analysis together with system level reverse synthesis can better serve for safe and secure development of the next generation e-voting systems.

Representational issues. As we noted, the modeling of assets and their flows can be done in two abstraction levels —that is, at higher level (using UML diagrams) and at lower level (using NuSMV specification). They have their own pros and cons. Although modeling at higher level easies and facilitates communication with non-domain users (as demonstrated in Chapter 3), our experience indicates that model extension at diagram level is cumbersome and has the issue of scalability —the model becomes too complex even with fewer number of assets, and this reduces the readability of the assets-flow models. In contrary, lower level extension of the model permits to perform complex analysis of the possible evolution of the assets in systematic way such as by exploring the push-button technology of model checking facilities. Well, naturally, this also suffers from a commonly known problem in model checking —i.e., the *state space explosion problem* (Clarke et al. 2000). Being aware of these issues helps choosing the right level of abstraction while modeling and specifying the assets-flow.

Finally, it must be clear that the construction of the extended model, whose generation can be automated, is currently performed by hand using the methodology and the translation strategies we described. The analysis approach we took, however, is very similar to that of FSAP/NuSMV-SA (Bozzano and Villafiorita 2007), for safety analysis, whereby a system specification is "enriched" with information about faults and analyzes are carried out to understand the effect and impact of faults on safety require-

ments expressed in the form of LTL/CTL formulae. Analogously to what happens in safety analysis when analyzing, e.g., the loss of a critical functions, enhancing the procedures results in reducing the probability of an attack or making the attack more complex, rather than eliminating it.

Chapter 5

Formal Analysis by Reverse Synthesis

In this chapter, we show how formal methods can be effectively used for the specification and verification of existing e-voting systems. Our aim is targeted to help voting machine designers and others verify critical security properties in DRE voting machines. We extend the methodology presented in the previous chapter to specify and verify also system level requirements. More specifically —by customizing the methodology presented in the previous chapter— we specify and verify one of the currently deployed e-voting systems named the Election Systems & Software (ES&S) electronic voting system. We call this approach reverse synthesis since the specification and the critical requirements are mainly derived from existing documentations of the deployed system.

Thus, after the introduction, we describe the relevant materials for the specification and analysis of the ES&S system: namely, the components of the ES&S system, voting process, critical requirements that the system must respect, as well as the known attack scenarios. We then present the main motivation of the reverse synthesis approach and the benefits it brings. Subsequently, we present the specification (using ASTRAL description language) and analysis (using the PVS analysis tool) of the ES&S

voting system, by first focusing on the nominal scenarios and followed by attack scenarios' specifications and analyzes.

5.1 Introduction

As we have mentioned several reports criticizing the trustworthiness of DRE-based voting machines, computer security researchers have not been able to alleviate citizens' concerns about this critical infrastructure. The researchers pointed out that e-voting systems often share critical failures in their design and implementation, which render their technical and procedural controls insufficient to guarantee trustworthy voting. Moreover, there are certain types of errors that are common in requirements specifications, and that have a negative effect on their usefulness for developing and testing such systems.

In California, these studies resulted in the Secretary of State allowing the use of e-voting machines only in special situations and with various changes to the electoral procedures. Several such changes shift the implementation of security requirements from e-voting systems to poll workers. For instance, (California Secretary of State 2007) states that

> *"no poll worker or other person may record the time at which or the order in which voters vote in a polling place."*

It is quite evident that a new generation of more carefully designed and engineered machines could move various checks currently performed by poll workers back to hardware and software. However, the success of this new generation of voting machines depends on our ability to capitalize from the lessons we learned using and analyzing the systems currently deployed, which we call a reverse synthesis approach.

To avoid confusion with other meanings, we use the following specific meaning throughout the chapter:

> *Reverse synthesis is a process of developing a high-level abstract specification for existing system and applying techniques to analyze whether the specification meets its critical security requirements.*

The specification development extracts an abstract specification from various sources that describe the behavior of the system and which is then to be used in the proof of implication with the security requirements.

Technical elements of our approach. We focus on how the use of formal techniques can help ensuring fair elections. More specifically, we derive formal specifications along with security critical requirements for ES&S system. The specification of the system and the critical requirements are mainly derived from various sources. The ultimate goal, therefore, would be a generic specification where some domain actors could easily convince themselves about the correctness of relevant security properties. In fact, it is a very hard task to convince about the full correctness of the system nor by no means we are claiming we solve this problem. Rather, we focus on the formal specification of the behaviors of the system and their verification against a subset of critical security properties. The specification and the verification of security properties, is a step towards fostering open specification and the (partial) verification of a voting machine. In addition, we specify attacks that have been shown to successfully compromise the system. With this information, we extend the original specification of the system and derive what we called the extended model. Using formal technique, we analyze the same critical requirements do indeed hold in the extended model.

Performing such synthesis is important for the following three reasons.

- First, it allows to discover some missing critical requirements for the specification and/or assumptions that were not met.

- Second, it allows to derive mitigation or countermeasure strategies when the system behaves differently than it should.
- Finally, not being able to prove the extended model would indicate that one, or more, of the threat actions violates at least one critical security requirement. This indicates that there must be an implementation error or an unsatisfied procedural assumption that results in the actual system or the environment not satisfying their respective formal specification.

5.2 The ES&S Electronic Voting Systems

A high-level overview of e-voting systems is given in Section 2.3.1. Here, we mainly focus on the description of the ES&S voting system components in relation to this chapter. We then describe (informally) a set of security critical requirements for these components individually, as well as the system as whole. The formal specification which will be shortly provided, therefore, is based on these information.

5.2.1 The System Components and Voting Process

5.2.1.1 The Voting System Components

Our discussion of the ES&S voting system components is based on what each component does, how each component exchanges input or output, and the underlying assumptions made to each component with respect to the specification stand point. More specifically, we focus on the following components of the ES&S voting system (see also Figure 5.1):

- **DRE.** Direct Recording Electronic voting machine, called the iVotronic. It is equipped with a touch-screen where the voter casts his or 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 (RTAL) is a continuous feed thermal printer manufactured by FutureLogic, Inc.[1] specifically for use in DRE voting systems. It performs the function of VVPAT on the ES&S DRE machines. It produces a paper-based record of the choices selected by the voter. The RTAL is connected to the DRE by a standard 9-pin RS232 serial cable, and mounted behind a transparent plexiglass window next to the DRE. The (paper) voter's choices are under this transparent glass so that they cannot be modified other than through the normal voting procedure. The main purpose of collecting the paper records for each eligible voter is for auditing.

- **PEB.** The Personalized Electronic Ballot (PEB) is a palm-sized device containing a PIC microcontroller, 2MB of flash storage, a bi-directional infrared (IR) transceiver, and battery. There are two kinds of PEBs identified by the read-only information burned in the PIC: their serial number, and more importantly by their PEB Kind. The two documented PEB Kinds are supervisor and voter, which are visually differentiated by a red and blue band in the casing respectively. It is 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.

- **CF Card.** Compact Flash cards are used to hold files too large to fit in the PEB flash storage as well as audio ballots and audit and results data. The ballot data is accessed by the DRE on demand, but the

[1]The FutureLogic, Inc http://www.futurelogic-inc.com/

presence of the CF card is checked periodically and the DRE will not boot without its presence. These cards are identified by their serial number. The card must be present to open and close the terminal. At poll closing, the audit data is automatically dumped into the card.

Figure 5.1: Main Components of the ES&S voting machine.

5.2.1.2 The Voting Process

The full election process involves many activities beyond what a poll worker and a voter typically experience in the polling station. Even if the exact processes differs depending on the specific voting technology in question, we distinguish, in particular, three major phases in the voting when using DRE-based machines (see also Figure 2.3): pre-electoral, electoral (during voting), and post-electoral phases. Before election day election officials use the election management system (EMS) to set up the election. In particular, the ballot definition files are prepared and loaded directly onto the DREs, CF cards are installed, and printers are assigned for each DRE machine. Moreover, the EQC is stored in the DRE so that the DRE can authenticate, upon the insertion, a qualified PEB.

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 DRE 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. The ES&S 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 (i.e., file) 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.

5.2.2 Informal Description of Critical Requirements

We describe a list of security properties that the system must respect. The security goal is that even in the presence of an adversary (see Chapter 4), the system should meet these properties. For instance, DRE should record the voter's intent exactly as the voter desires. Further, an adversary should not be able to undetectably alter the votes once they have been successfully stored. We wish to specify these kinds of properties and validate against the system model, as well as in the presence of threat actions corresponding to each attack scenario, which will be discussed subsequently.

A number of requirements that the ES&S system must satisfy are enumerated in the ES&S system manual (Inc. ES&S 2007) (such as configuration instructions and the user's manual) and a corresponding video[2], which describes how the system works on election day. Instead of describing prop-

[2]http://www.essvote.com/HTML/voter_outreach/ivotronic_flash.html

erties such as like (Sastry 2007) (e.g., "A ballot cannot be cast without the voter's consent to cast it; the DRE only stores ballots that have been confirmed by the voter.") or like (McGaley 2008) (e.g., "The e-voting system shall be protected against threats to its availability including: malfunction, breakdown and denial of service attacks."), we rearrange and split the properties so that providing their equivalent formal specification is manageable. In fact, the rearrangement we followed is the one we described in (Villafiorita et al. 2009a). It should be clear that we are not claiming we provide new requirement elicitation techniques nor structuring approaches.

In the following, we list the most important critical security requirements that the ES&S voting system must meet. The list is by no means exhaustive, but is chosen to reflect important properties that are essential building blocks for most DRE e-voting machines equipped with VVPAT. Notice that the presentation of the requirements does not follow any order of importance nor in sequel.

Requirements for the DRE.

A correctly functioning DRE must satisfy the following properties.

Property 1 *The same CF card must be present throughout the voting session;*

Property 2 *The RTAL must be connected to the DRE throughout the voting session;*

Property 3 *The DRE must authenticate the PEB using the EQC, and the same master PEB must be used to open and close the terminal;*

Property 4 *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;*

Property 5 *The ballot presented to the voter must be complete;*

Property 6 *Display screens presented to the voter must accurately reflect the ballot downloaded from the PEB and the selections made by the voters;*

Property 7 *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;*

Property 8 *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;*

Property 9 *The DRE must automatically forbid an overvote;*

Property 10 *The DRE must report undervoted races, if they exist, and the review screen must display the message "BALLOT NOT COMPLETED";*

Property 11 *When the voter confirms his/her ballot, the ballot images recorded in the local storage must correctly reflect the selections made by the voter;*

In other words, Property 11 states the fact that the DRE must not change the ballot after the voter chooses their candidates.

Property 12 *The DRE terminal should start chirping if there is no input from the voter for 10 time units since the last input but not after he/she confirmed.*

Requirements for the RTAL.

The RTAL must satisfy the following properties:

Property 13 *The RTAL should scroll up a minimum distance after the summary has printed, in order to out of sight the previous vote;*

Property 14 *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.*

Even if Property 6 and Property 14 state different requirements, they are meant to express the fact that the voter must have a chance to preview

(both on the DRE screen and on the RTAL window) the contents of the ballot and accept or reject it.

Requirements for the PEB.

The PEB must satisfy the following properties:

> **Property 15** *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;*

> **Property 16** *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:

> **Property 17** *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:

> **Property 18** *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 that were accumulated and reported by the DRE and RTAL.*

The above requirement can further be classified into the following requirements.

> **Property 18. 1** *The vote entries printed on the RTAL tape during and after the election must be equal to the cast ballot records plus the rejected vote in the DRE;*

Property 18. 2 *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;*

Property 18. 3 *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.*

Property 18. 4 *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;*

We want to remark that some properties documented in the ES&S election day checklist manual —such as, while downloading the election results from the DRE after the election is closed, the PEB should not be removed until the download finishes and safe to remove it— are not intrinsic to the system functionality instead they are either procedural or environmental assumptions. To make a reasonable and, of course, provable specification, however, one can make assumptions about those facts.

5.2.3 Selected Attack Scenarios

Next, we give a short overview of selected attack scenarios that are discussed for the ES&S system in the EVEREST report. We assume that voters can leave the voting booth without checking the votes shown on the confirmation screen and/or on the RTAL screen do indeed accurately reflect their intentions. This assumption, in fact, is based on what happened in real scenario, i.e., even if the ES&S system offers (in general, any DRE-based machines) voters the opportunity to verify their vote, some voters leave the voting booth without completing the voting procedure[3].

[3]However, in ProVotE e-voting system scenario #3 and scenario #4 may be detected easily by the poll worker since the system offers a signaling system, which is installed outside the voting booth to help assisting the poll worker whether the voter completed the voting procedure or not (see in (Villafiorita et al. 2009b)).

Moreover, like any other voting systems, the ES&S voting system can be subject to attack by a number of different types of attackers with different capabilities. An attacker can be *outsiders* (have no special access to any of the voting equipment), *voters* (have limited and partially supervised access to voting systems during the process of casting their votes), *poll workers* (have extensive access to polling place equipment), *election officials* (have extensive access both to the back-end election management systems and voting equipment), and etc.

We now present four selected attack scenarios for which we give formal specification later. A sequence diagram for each corresponding attack scenario is also sketched.

1) Changing the vote for an unattentive voter. In this scenario, the voter proceeds with the normal voting process and the attacker intercepts the process just before the review ballot is displayed. The attacker steals votes by assigning them to the candidate who s/he desires to win. The modified vote is displayed on the DRE review screen and the change is printed on the RTAL tape. Because the voter is unattentive, s/he will not look at the screen nor the RTAL. However, if the voter does check the screen or the printed output and discovers that an error has been made, s/he can recast the vote and the attacker will stop stealing votes for a period of time. Otherwise, the attacker's modification is stored locally upon the voter's confirmation (See Figure 5.2).

2) Changing the vote for a careful voter. This scenario assumes the voters carefully cast, check the screen and printout, and confirm. However, they are not familiar with all the details of how their votes are printed on the RTAL tape. The attacker does not intercept the normal voting process until after the cast ballot and confirmation screens have been shown to the voter. At this point, the attacker changes the voter's electronic ballot, and the RTAL prints the modified selection. The RTAL then immediately

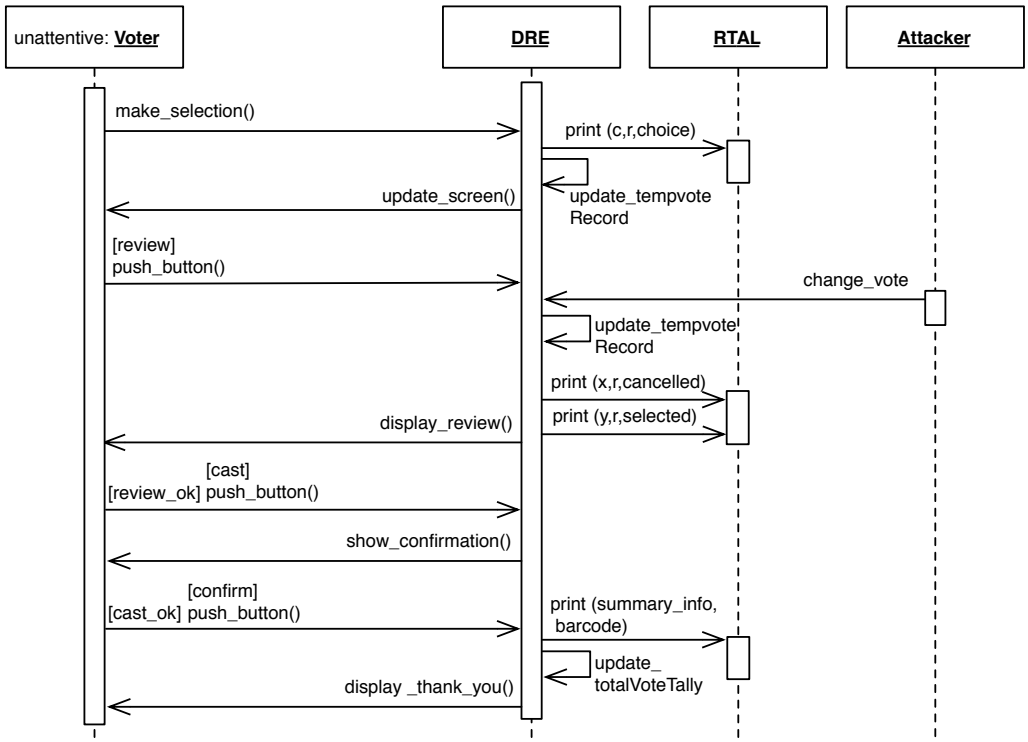


Figure 5.2: Changing an Unattentive Voter's Vote.

prints the summary information along with the barcode.

3) Canceling or completing the vote for a fleeing voter. In this scenario the attacker takes advantage of a fleeing voter, a voter who does not complete the voting procedure, by intercepting the call to the routine that enables a chirping sound, which alerts the poll worker that a voter has fled. There are two possible scenarios depending on the voter's vote:

1. If the fleeing voter voted against the attacker's candidate, then the attacker does nothing and lets the chirping routine perform as it should. The poll worker then discards[4] the ballot and there will be one less vote for the undesired candidate.

2. If the fleeing voter voted for the attacker's candidate but s/he did not

[4]In Ohio the votes of fleeing voters are discarded. In California, whereas the poll worker casts these votes.

complete the voting process then the attacker completes the voting process. This results in another vote being cast for the attacker's candidate.

4) Faking a fleeing voter to cancel a vote. This attack scenario is similar to #3. However, in this case the attacker cancels the vote by making it looks like the voter fled. In particular, if the voter did not choose the candidate that the attacker wants, the attacker intercepts the confirmation process and pretends to cast the ballot: the normal "thank you" screen is displayed, but nothing is printed on the RTAL tape. After some amount of time elapses (during which the voter most likely leaves the voting booth) the attacker directs the system to display the confirmation screen. Then after another reasonable amount of time has passed the attacker calls the chirping sound routine and the machine immediately starts chirping. A poll worker will think the voter was a fleeing voter and the ballot will be discarded (see Figure 5.3).

In the above attack scenarios, various low level details that are not the interest of formal specification and verification are omitted. All the descriptions we give are completely from view point of formal methods. Their corresponding formal specifications will shortly be discussed in terms of threat actions. Moreover, these are not the only attack scenarios for ES&S voting system. Unfortunately, not all are interesting to specify and/or analyze against the model under analysis. In the contrary, for some attack scenarios we cannot provide their corresponding formal specifications, as they are too low-level to specify and are implementation specific in some cases.

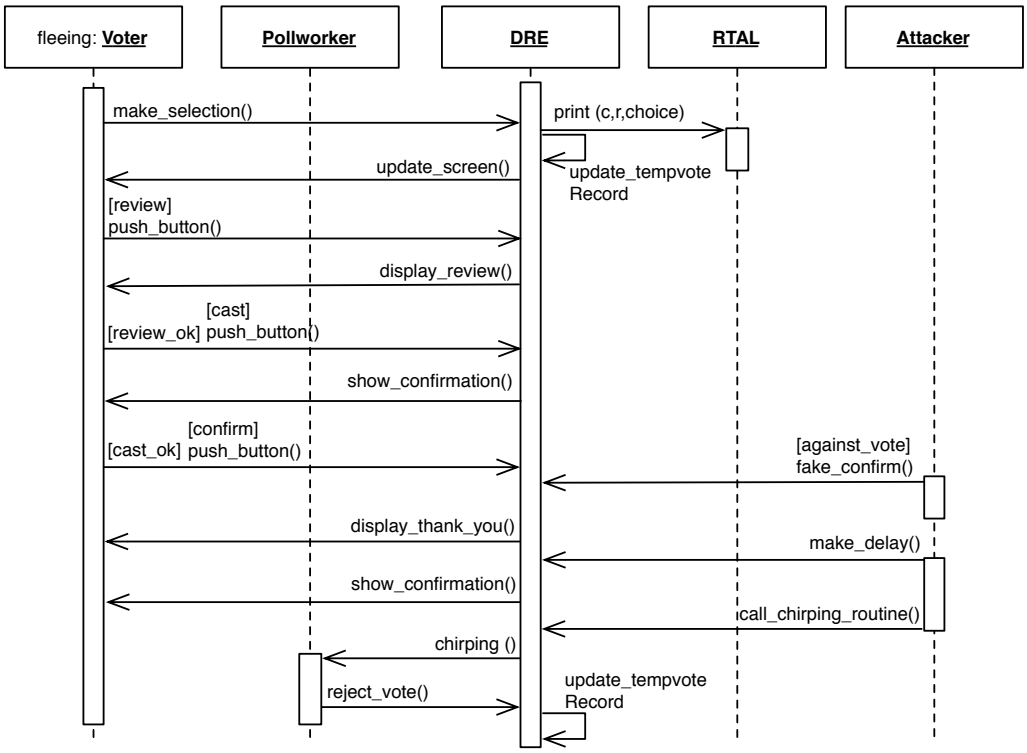


Figure 5.3: Canceling a Vote by Faking a Fleeing Voter.

5.3 Reverse Synthesis Approach

Fairness and security of electronic elections depend upon a careful allocation of requirements to the procedures and to the systems used. In fact, 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 other fundamental properties that the procedures can only partially assure. 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, given that the poll workers follow the prescribed procedures.

Analyzing such requirements and their allocation 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 different allocation of requirements between systems and procedures could improve the overall security of the election process so that we can build the next generation of e-voting machines.

However, in order to achieve these goals, we need an approach that allows us to easily experiment and reason about, for example, different allocations of requirements. At the same time, it has to be precise and exhaustive, so that security and dependability consequences of any specific choice are highlighted. Formal techniques clearly fit both needs. Our goal in this chapter is not to show end-to-end verification nor development of the e-voting system using formal methods. However, we wish to demonstrate how their use can help ensuring fair elections.

Figure 5.4 depicts our approach for the reverse synthesis process. In the figure, the *"nominal"* behavior refers to all the intended operations of the system under analysis. By contrast, the *"non-nominal"* is meant to describe those behaviors of the system that deviate from intended operations of the system due to attack actions. More specifically, we derive formal specifications along with critical security requirements for the ES&S system. The specification of the system and the security critical requirements are mainly derived from available information sources: the EVEREST report (McDaniel et al. 2007), the ES&S election day checklist and user's manual (Inc. ES&S 2007), a video[5] that shows how the ES&S system

[5]http://www.essvote.com/HTML/voter_outreach/ivotronic_flash.html

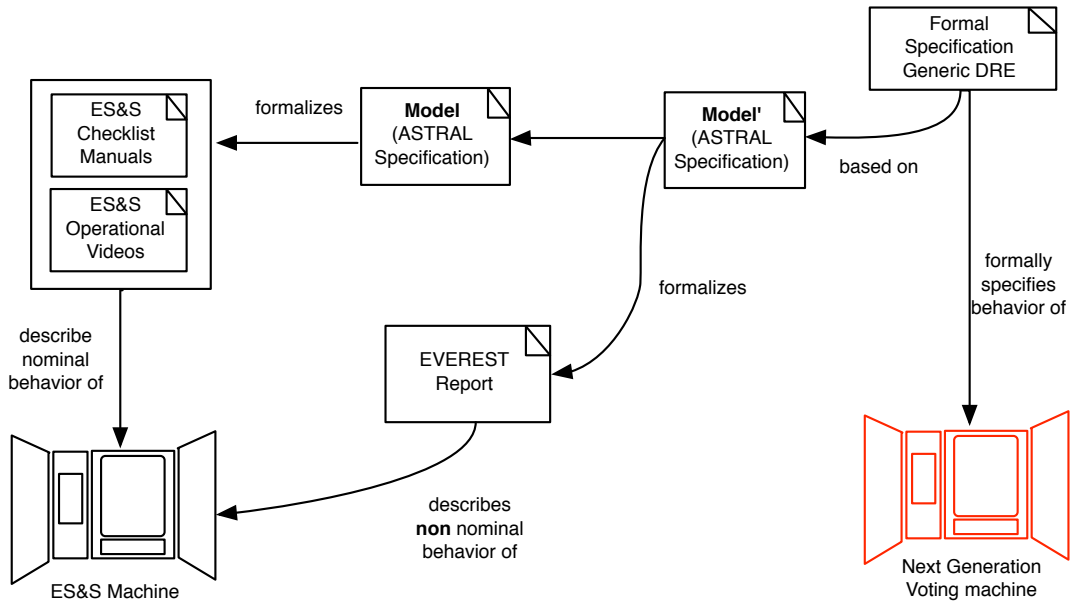


Figure 5.4: The approach of Formal Specification and Verification by Reverse Synthesis.

works on election day, and other requirements suggested in the literature (e.g., (Mercuri 2001, Volkamer and McGaley 2007, Sastry 2007, McGaley 2008)). Using formal analysis tools, we can assess the strengths and weaknesses of the system. Results or feedback gained from the formal analysis can be a basis for specifying and analyzing generic requirements from which the next generation of voting machines can be built. The analysis tools we use are the proof guidance strategies by Kolano (1999) and the SRI PVS analysis tool (Owre et al. 1993, SRI).

Moreover, we believe that besides analyzing the system against its requirements, it is equally important to perform analysis under malicious circumstances where the system execution model is enriched with attack behavior. Notice that the first model we build for the ES&S system only specifies the intended behaviors of the system —i.e., *Model* in Figure 5.4. Therefore, we should specify and extend the model with some generic attacks that may defeat some behaviors of the system, and analyze the resulting model against the security properties. A successful analysis on the

resulting model (i.e., *Model'* in Figure 5.4) can reveal important information about the system, which in turn helps detecting missing requirements or unwarranted assumptions about the specification we developed. In addition, this allows us to sketch countermeasure strategies to be used when the system behaves differently than it should and to build confidence about the system under development.

5.4 Overview of the ASTRAL language

ASTRAL (Kolano et al. 1999) 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.

5.5 Formal Analysis of an e-voting System

As noted previously, e-voting systems resemble a real-time system which consists of various components all working together aiming to run a correct and trustworthy election. We turn our attention in this section in specifying each individual component of the voting machine and their communication for enabling formal analysis. In order to do this, we need to formalize their behaviors and properties that we wish to prove. Thereafter, first, verify that each individual component must meet its specification when considered separately; second, analyze what happen when these components are all together —namely, the resulting machine must satisfy certain critical

properties that we would expect a correct voting machine to meet.

We now present the specification and verification of the ES&S voting machine by showing a sampling of the specification that, we believe, provides the flavor of the work. To make our strategy repeatable, we follow the following generic methodology; it is a variant of the methodology we devised for procedural security analysis in Chapter 4.

- *Specify the Voting Process.* During which we specify all the nominal behaviors corresponding to each component of the machine as we discussed in Section 5.2.1.1 using the ASTRAL specification language;
- *Specify Critical Requirement.* During which we specify properties listed in Section 5.2.2, procedural requirements, as well as environmental assumptions about the system operation;
- *Perform Formal Analysis.* During which we validate the specification and generate proof obligations for the properties using the ASTRAL SDE. The proof obligations are then analyzed using the PVS analysis tool;
- *Specify Attacks.* During which we extend the nominal specification of the system with attack scenarios discussed in Section 5.2.3. For each attack scenario, we specify a sequence of threat actions which each corresponds to a transition specification in ASTRAL;
- Repeat the analysis and analyze results.

Even if it depends on the kind of analysis we wish to do, it is sufficient to say that the above methodology can be repeated until suitable results obtained.

To make the specification and analysis simple but without losing generality, one DRE machine per polling station is assumed. We assume also

there is one CF Card, one RTAL, and one PEB (either master or supervisor) per DRE machine used in the election; moreover, we assume there is one race per screen. We do not specify and verify activities related to pre-electoral phase. Instead, if relevant, we make assumptions about these activities in the initial assertion of the specification that these operations have been carefully and correctly done at the central location —i.e, at the electoral central.

It is also worth remarking that we will not specify nor analyze usability requirements, such as "the voting device must not display any information about the voter's selections outside the vote casting interface; the vote casting interface must clearly indicate to the voter whether the voting device is in an active state or an inactive state." However, we consider specifying the possible states of the machine (e.g., the machine is in the *poll worker*, *voter*, *sleep* or *chirping* mode), since such information help us understanding the different operations that a poll worker or a voter experiences when interacting with the machine.

5.5.1 Specification of the ES&S Voting Process

We formulate each component of the ES&S voting system as an ASTRAL process instance (See Figure 5.5). There is a process specification for each process type declared in the global specification —i.e., four process types are declared in the global specification of the ASTRAL model of the ES&S system.

```
PROCESSES
  the_DREs: array [ 1..Number_Of_DRE ] of DRE_Process,
  the_RTALs: array [ 1..Number_Of_RTAL ] of RTAL_Process,
  the_PBEs: array [ 1..Number_Of_PEB ]  of PEB_Process,
  the_CFCards: array [1..Number_Of_Card]  of CFCard_Process
```

The above declaration indicates that there are `number_Of_DRE` DRE in-

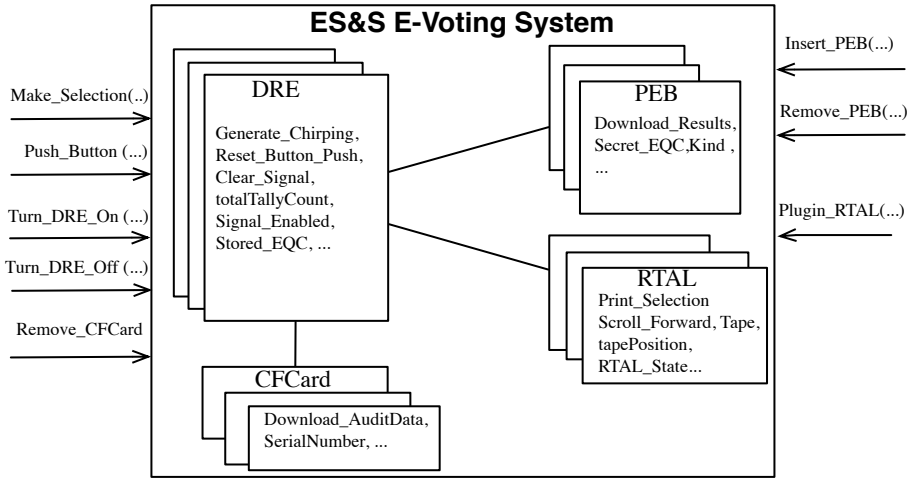


Figure 5.5: A simplified view of ES&S voting system

stances of type `DRE_Process` in the model, where `number_Of_DRE` is a global positive integer constant declared in the constant declaration part, and similarly for the other components.

We defined user defined types and constants to represent useful concerns about the ES&S system inputs and outputs, like in the following snippet specification:

```
TYPE
    DRE_ID: TYPEDEF p: ID ( IDTYPE( p ) = DRE_Process ),
    PrintValue,     /* unspecified type*/
    Title IS SUBTYPE OF String,
    Candidate_Name IS SUBTYPE of String,
    DecisionType:  ( Selected, Canceled ),
    Button:  ( RESET, EXIT, CANCEL, CLOSE, _START, NEXT,
               BACK, REVIEW,  CAST, CONFIRM ) ,
  [...]

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

The DRE_ID line declares DRE_IDs to be exactly those ids that are process instances of type DRE_Process. The DecisionType and Button are enumerations, that represent, respectively, the voter's decision on a candidate for a given contest and the buttons that can be used to interact with the touchscreen. In contrast, the first two constants associate each DRE with a unique CF Card and RTAL printer, which take DRE_ID as an argument and return CFCard_ID and RTAL_ID respectively. Make_Print_VoteEntry represents the print format on the RTAL paper tape when the voter selects or cancels a particular candidate. In addition to printing vote selection, the RTAL also prints start and summary information for each voting session.

5.5.1.1 Modeling the DRE Process

We now discuss the DRE_Process specification in detail. The below initial clause of the DRE model states that a CF 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
               & CFCard_Installed = TRUE
               & CFCardSerialNumber =
                  Which_CFCard_Installed.SerialNumber )
  & EXISTS rt: RTAL_ID
    ( rt = Plugged_In_RTAL ( Self )
          ->   Which_RTAL_Plugged_In = rt
             & RTAL_Plugged_In = TRUE )
```

Using the import clause in the interface section of DRE_Process, the process can import globally declared types, constants, and definitions, as well as variables and transitions exported by other processes in the system. For instance, Installed_CFCard and Plugged_In_RTAL are constants de-

clared in the global specification and are imported using the import clause of the `DRE_Process` process. Similarly, the process can also export variables and transitions which can be used by other processes. For instance, `Which_CFCard_Installed` below is an exported variable.

```
VARIABLE
    NumberOfSelected ( Race_Num ) : Non_Negative,
    totalTallyCount ( Candidate_Name, Title ) : Non_Negative,
    Which_CFCard_Installed: CFCard_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 fist two variables above.

The communication between the DRE and the RTAL processes is modeled by the variables:

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

where the first variable signals that the DRE is sending information to the RTAL 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 mode of the terminal during election day. We use the following variables:

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

that indicate, respectively, the phase of the election (pre-voting, during voting, and post-voting phases), the terminal mode (Figure 5.6), and the state of the poll (opening, opened, closing, or closed). The last two variables are only meaningful during the actual election day —i.e., `Which_Phase = During_Voting`.

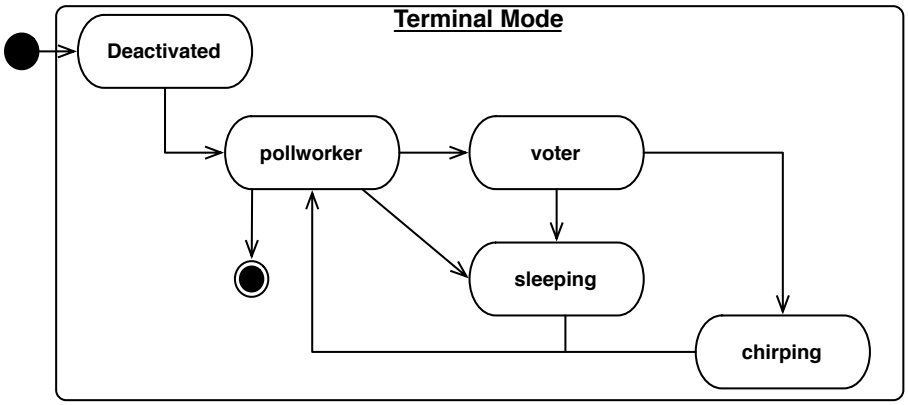


Figure 5.6: The possible states of the DRE terminal mode.

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 (such as `NEXT` or `BACK`). We model such interaction by assigning an integer number to each screen shown to the voter and by defining a function that takes as input a screen number and returns the information to be displayed and the buttons available. The variable `Display` of type screen, is used to hold the state of the screen as it is to be shown to the voter while s/he is voting. For example, 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.

Once we capture the relevant data structures that allow to hold information about the DRE, the next step is modeling the behavior of the DRE itself using ASTRAL transition specifications. Namely, each operation that triggers behavioral change to DRE is encoded as a transition in ASTRAL.

An example of such operations is the insertion of PEB into the DRE by a poll worker changes the current state of the DRE to some other state.

The following code models the activation of the machine for the election:

```
TRANSITION Turn_DRE_On
   ENTRY [ TIME : T_D_On_Dur ]
     Terminal_Mode = Deactivated
     & DRE_State = Initial_State
     & Which_Phase = Pre_Voting
     & EXISTS cf: CFCard_ID
       ( cf.SerialNumber = Stored_CFCardSerialNumber
         & Which_CFCard_Installed = cf & CFCard_Installed)
     & ~MachineTurnedOn
    EXIT
     MachineTurnedOn
```

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

In contrast, the transition

```
TRANSITION Turn_DRE_Off
   ENTRY [ TIME : T_D_On_Dur ]
       Which_Phase = Post_Voting
       & Terminal_Mode = Deactivated
       & EXISTS card: CFCard_ID
            ( card.SerialNumber = Stored_CFCardSerialNumber
              & Which_CFCard_Installed = card )
       & CFCard_Installed
       & MachineTurnedOn
   EXIT
```

```
            ~MachineTurnedOn
```

models the effect of turning the machine off.

An ES&S DRE requires a poll worker to insert a qualified PEB device in order to allow various operations to run the election. These operations include loading the appropriate ballot, opening or closing polls, initializing the ballot, collecting election results, and performing various administrative tasks. We modeled all these aspects with appropriate transitions.

The following snippet specification encodes the ballot loading operation prior to start election.

```
TRANSITION Insert_PEB ( p: PEB_ID )
  ENTRY           [ TIME : I_P_Dur1 ]
    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 )
 EXIT
     Which_PEB_Inserted = p & PEB_Inserted
     & ( FORALL R: Race
       ( Race_Candidates ( R ) = P.Candidates_Of_Race ( R ) ) )
     & Ballot_Loaded
 [...]
```

Once the ballot is loaded while inserting the PEB, the poll worker must remove the inserted PEB safely. This is done by calling `Remove_PEB` transition. The result of the remove operation, in the nominal case, is changing the state of the system to allow voting and putting the machine in sleeping mode (i.e., `Which_Phase = during_voting`, `DRE_State = Opened`, and

`Terminal_Mode = sleep_mode`). In other words, this indicates that it is now voting time, the poll is opened for election, and the terminal mode goes to sleep.

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 = pollworker
       & (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 ) ) = FALSE )
       & FORALL R: Race ( Number_Of_Selected ( R ) = 0 )
       & Ballot_Initialized & ~Proceed_Ballot_Init
       & underVotedRaces = EMPTY
```

The entry conditions specify that the poll has to be opened in the poll worker terminal mode, the PEB is inserted, and the ballot has not been initialized for the voter who is ready to cast her/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 all the variables values from the last voter are reset —i.e., the `Picked` value for each *candidate-race* pair, 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 variable `Ballot_Initialized` is set to true.

There are four nominal situations in which the PEB can be removed:

1. after the poll worker loaded ballots prior to opening the DRE terminal for voting;
2. after the poll worker initialized the ballot for the next voter during the voting phase;
3. after the poll worker performed administrative operations (such as after correcting the chirping terminal mode) during the voting phase;
4. after the poll worker downloaded the election results after the terminal is closed for election.

If the `Remove_PEB` transition has been fired because the poll worker initialized (see below) the ballot, then the terminal mode changes to voter mode, the current screen becomes the starting screen for the eligible voter with a `START` button on it.

```
[...]
/*Removing the PBE after the ballot has been initialized for the voter.*/
    Terminal_Mode = voter_mode & scrName = START_SCREEN & scrNumber = 0
    & Screen_Buttons ( scrNumber )  BECOMES   START_BUTTON
    & Min_Display ( scrNumber )  BECOMES Display_Info
              ( Push_Start_Button_To_Start_Voting,
                Screen_Buttons ( scrNumber )  )
[...]
```

In a touchscreen 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. Noticed that we assume one race per screen. In reality, however, a screen can display more than one race. As noted before, we assigned an id (as numeric number and name) to each screen. Our specification can be extended for multiple

races per screen, e.g., by introducing a variable with two parameters one related to the screen number and the other related to a race id. However, our assumption of one race per screen is sufficient for the analyses we wish to perform.

We introduce an exported transition called `Make_Selection`, which must be called from external model. This transition captures the facts described above. With this transition, the voter first interacts with the screen that presents the ballot choices (by using `Push_Button` transition as shown below). After making their selections, control flow passes to the screen upon calling `Push_Button` transition that performs a limited role: presenting the voter's prior selections and then waiting for the voter to either choose to modify their selections, or choose to cast and then to confirm their ballot.

```
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)
    & Display( scrNumber ) =
       Display_Contest ( Race_Title ( currentRace ),
              Displayed_Candidates(currentRace),
              Screen_Buttons ( scrNumber ) )
    & EXISTS C: Candidate
       ( C ISIN Displayed_Candidates (currentRace)
         & Candidate_Name ( C )  = cName )
    & ~Signal_Enabled
```

The above code 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. We used `Picked` variable 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

```
EXIT
   IF ~Picked' ( cName,Race_Title (currentRace') )  THEN
     IF Number_Of_Selected' ( currentRace' ) + 1
          <= Max_Choice_Per_Race ( currentRace )
     THEN  /*over-vote is not attempted.*/
       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 )
       /*set variable value for the RTAL to print.*/
       & pickedName = cName & pickedValue = Selected
       & Signal_Enabled & Which_Signal = Vote_Signal
       & currentRace = currentRace'
     ELSE  /*else over-vote is attempted.*/
       Min_Display ( scrNumber' )  BECOMES
          Display_Info (OverVote_Prohibited, NoButton)
     FI
 ELSE  /*else, cancel the previous choice.*/
    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 )
    [...]
```

There are two possible cases when a voter marks a candidate on the screen:

1. *Making a selection.* In this case, 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 requests 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.

2. *Canceling a previous selection.* In this case, 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 and cancellation expressed in these exported variables information are sent to the RTAL.

A voter and a poll worker interact with the screen while casting votes and administrating the election. The encoding of how the DRE handles these button pushes is very difficult, as it requires to understand the various information flow corresponding to each button push activity. We encoded all the logics related to each button push by introducing a transition called Push_Button (b : Button), which is an exported transition. It has a num-

ber of entry/exit pairs corresponding each button. Therefore, for each button we defined the corresponding entry/exit pair.

For instance, the entry and exit assertions that corresponds to the START button are as follows:

```
ENTRY
  /*Case1: Start button.*/
    b = START_BUTTON & b ISIN Screen_Buttons ( scrNumber )
    & scrName = START_SCREEN & scrNumber = 0
    & ~Button_Pushed ( b ) & Which_Phase = During_Voting
    & Terminal_Mode = voter_mode & ~Signal_Enabled
EXIT
    Button_Pushed ( b ) BECOMES TRUE & scrNumber = 1
    & currentRace = Which_Race ( scrNumber )
    & Screen_Buttons ( scrNumber ) BECOMES  NEXT
    & Display ( scrNumber ) BECOMES Display_Contest (
           Race_Title ( currentRace ),
           Displayed_Candidates' ( currentRace ),
           Screen_Buttons ( scrNumber ) )
    & voterNumber = voterNumber' + 1
    & ~Ballot_Initialized
/* Make available for RTAL to print. */
    & RTALMessage = VOTE_SESSION_STARTED
    & Signal_Enabled & Which_Signal = Start_Signal
```

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 should be in the screen button list for the current screen, the current screen is START_SCREEN, the corresponding screen number equals zero, and the button is not 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. In contrast, the exit assertion indicates that the voter has pushed START button, the screen number is incremented by one, the current screen displays the first race, and the only

button available to push is NEXT, the number of voters who visited the poll is incremented by one, and since the voter is already started voting, thus, the Ballot_Initialize is reset to false. In addition, the DRE updates the value of Signal_Enabled, Which_Signal, 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 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 individual button press can be different depending on which button was pushed.

After the voter has completed all of his/her votes, the voter has to cast and confirm his/her 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 (by using the BACK button) and making any changes before casting their ballot. Pressing the CONFIRM button will cast the ballot.

```
[...]
 /* Store the vote locally because the voter has confirmed.*/
   FORALL C: Candidate, R: Race
     ( C ISIN Displayed_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'
      & underVotedRaces = underVotedRaces'
      & RTALMessage = BALLOT_ACCEPTED_UNDERVOTE
     ELSE
      RTALMessage = BALLOT_ACCEPTED
      & underVotedRaces = NoUnderVotedRace
    FI)
  [...]
```

In addition to updating the total tally, the DRE keeps track of log data (such as undervotes races, if they exist, in `underVotedRaces`).

When a voter flees (it leaves without confirming the vote), the iVotronic makes a chirping sound after reasonable time, which alerts a poll worker to attend the terminal. This is captured by the following transition

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

In the ES&S voting system, the DRE is also responsible for clearing the previous signal value and the button push, by performing the transitions `Clear_Signal` and `Clear_Button_Push`, respectively. We omit the discussion of the remaining transitions since they follow more or less similar pattern with the transitions discussed so far.

5.5.1.2 Modeling the RTAL Process

The RTAL collects the output sent by DRE internally, mostly for auditing purposes. Namely, it prints vote actions exported by the DRE on a paper tape. In our specification, the RTAL is specified by an instance of type `RTAL_Process`. The paper tape contains a list of voter records, where each individual voter record is a continuous sequence of voter actions.

These are captured by the following variables.

```
VARIABLE
   Tape ( Pos_Integer ) : PrintValue,
   tapePosition: Tape_Number,  /*positive integer*/
   RTAL_State: RTALState,
   summaryPrinted: Boolean,
   VoteStartPosition ( Voter_Number ) : Tape_Number,
   VoteEndPosition ( Voter_Number )  : Tape_Number
```

The `Tape` variable 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. Moreover, the variables `VoteStartPosition` and `VoteEndPosition` delineate the voter record on the paper tape. In fact, the model of the RTAL process is similar as an array of continuous values of votes. Each time a voter makes a choice the corresponding record is inserted into the array and at the end of each vote confirmation empty values are appended to represent the rewinding operation of the RTAL.

Along the above line, there are two main behaviors of RTAL that are of

interest of specification: printing actions and rewinding the printer tape for secrecy reason, after printing the summary information on the tape. The former is modeled by `Print_Selection` transition. The latter, whereas is modeled by `Scroll_Forward` transition.

```
TRANSITION Print_Selection
    ENTRY           [ TIME : P_S_Dur ]
       My_DRE.Plugged_In
       & My_DRE.Signal_Enabled
       & RTAL_State = Wait
       & My_DRE.Which_Signal ~= NoSignal
```

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 RTAL is signaling to print.

```
(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
         [...]
        ELSE  /*voting entry printing*/
                Tape ( tapePosition )  BECOMES
                  Make_Print_VoteEntry ( My_DRE.pickedName,
                         My_DRE.currentRace, My_DRE.pickedValue )
          FI )
   ELSE  /*Summary printing*/
     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 = TRUE
FI)
```

After the transition is fired, depending on what signaling mode has been received by the RTAL, the corresponding entry is printed. Notice that each vote record is uniquely identified by a `barcode`, which encodes the voter's ballot selections in the RTAL record without revealing the identity of the voter. This `barcode` is printed on the tape along with the summary information of the vote entry. Upon the completion of printing the summary information, the printer is also scrolled forward by calling `Scroll_Forward` transition.

5.5.1.3 Modeling the PEB and CF Card Processes

The PEB device is specified by an instance of type `PEB_Process`. As mentioned earlier, the PEB device —in addition to 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— is used to transfer election specific data between Election Central and poll locations, which

is encoded by the variables `Candidates_Of_Race`, `tabulatedData`, and `copyOfBallotImages`.

We mentioned that according to the ES&S voting process specification, the DRE authenticates each PEB by its four digit EQC code (encoded by `Secret_EQC` variable). 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 our specification, we only use PEB kinds to distinguish PEBs (i.e., `Kind : PEBKind`).

The most important aspect to specify about the behavior 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. This is specified by the transition `Download_Results`.

```
[...]
 /*Download the election result */
   FORALL C: Candidate, R: Race
    ( C ISIN D.Race_Candidates ( R )
       -> tabulatedData ( C, R, D ) = D.TotalTallyCount ( C, R ) )
  &
 /*Dump copy of ballot images into this PEB. */
 copyOfBallotImages ( D )  BECOMES Download_BallotImage (
   { SETDEF Pair: Race_Candidates_Pair (
         EXISTS R: Race ( Pair [ Contest ]  = R
            & Pair [ Nominees ] = D.Race_Candidates ( R ) ) ) } )
[...]
```

On the other hand, the CF 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 undervoted races and the number of fleeing voters, indicated by the following variables:

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

We mentioned that there is one CF card per DRE machine used in the election. This card is uniquely identified by its serial number —i.e., `SerialNumber: Digit_List`. When the polls are closed, an audit file is saved automatically to the CF Card. In other words, upon closing the terminal while the master PEB is inserted, the DRE automatically enables the CF card to save audit data. From a formal specification point of view, however, we are only interested in the audit log file, which contains the undervoted races and the number of fleeing voters. This activity is modeled by `Download_AuditData` transition.

In summary, the complete ASTRAL specification of the ES&S voting process is approximately 1500 lines (33 pages long). We skipped a number of descriptions about the specification of each component[6].

5.5.2 Critical Security Requirements

Once we modeled the components of the ES&S system and their evolution, we need to model what critical requirements each individual component and the system as whole should meet, given that the assumptions about the behavior of the system and the external environment that interacts with the system. In particular, we specify two classes of requirements about the ES&S voting system: environmental or procedural assumptions and security requirements.

[6] The complete ASTRAL specification of the ES&S voting machine can be downloaded at `http://ict4g.fbk.eu/people/sisai/specs/astral-specs/ESandS_Original.spec`

5.5.2.1 Environment Requirements

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 control ES&S voting 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 then 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, e.g., the poll worker has to wait some amount of time to remove the PEB after loading the ballot, or after activating the ballot for the next active voter. However, they are equally important to carry out a correct and secure election. Therefore, we need to express these concerns in order to guarantee the critical requirements that the system should meet.

Next, we discuss some of the assumptions we made for the ES&S specification. We assume that a voter will pause for some amount of time between subsequent screen interactions. We do not make any assumption about the sequence of user actions, since it is very difficult to predict the user's action while s/he interacts with the DRE. 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 notifies him or her to remove the PEB.

We specified a number of environment assumptions for each components. For example, we specified ten such assumptions for the DRE under the

process type DRE_Process.

```
ENVIRONMENT
  /*min_pause is the minimum time between two subsequent selections */
    ( FORALL t: time
       ( Call [ 2 ]  ( Make_Selection, t )
           ->   Call ( Make_Selection )  - t > min_pause ) )
    &
  /*min_pause is the minimum time between two subsequent button pushes */
    ( FORALL t: time
        ( Call [ 2 ]  ( f, t )
          ->   Call ( Push_Button )  - t > min_pause ) )
     &
  /*Remove_PEB will be called after the ballot is loaded into
  the DRE and Notify_Time1 units has elapsed. */
    ( EXISTS t: Time, p: PEB_ID
       ( Now >= t + Notify_Time1 & p = past ( Which_PEB_Inserted, t )
         & past ( PEB_Inserted, t ) & past ( Ballot_Loaded, t )
         & past ( p.Kind, t )  = Master
         & past ( DRE_State, t )  = Opening
         ->   Call ( Remove_PEB, t + Notify_Time1 ) ) )
```

For instance, the above clause for DRE process states that there must be a minimum pause between two subsequent selections and button pushes. It also specifies the fact that the poll worker should only remove the PEB after the loading operation has passed and Notify_Time1 has elapsed. All these facts are important to proof the critical requirements, in particular requirements that involve exported transitions.

5.5.2.2 Security Requirements

In the ES&S voting system specification discussed earlier, the DRE must authenticate each PEB using its four digit election qualification code, 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 periodically check for the existence of the CF card. The RTAL printer should also be continuously working during the election period. All these activities are essential 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 EMS and the hardware used. 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 we are interested in are mainly concentrate on the integrity of election results.

We formulated each of the critical property from Section 5.2.2 as ASTRAL invariants, constraints, or schedules formulas. We now present examples of critical requirements specification, mostly related to the integrity of election results.

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` (Property 1):

```
( Change ( CFCard_Installed, Now )
 & ~CFCard_Installed
  ->  FORALL t: Time
        ( t >= 0 & t < Now
        & past ( Stored_CFCardSerialNumber, 0 )  =
           past ( Stored_CFCardSerialNumber, t )
        & past ( CFCard_Installed, t )  )  )
```

Specifying that all voter selects/cancels are displayed on the DRE screen

(Property 5 and 6): The following constraint expresses the fact that when a voter selects or cancels a candidate C for a given race R, the DRE screen must be updated accordingly:

```
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))
```

A local constraint 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 (Property 9):

```
( 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 fact that a DRE is chirping indicates that at least ten units have passed since the last ballot activity. This is expressed by the following local schedule requirement of the `DRE_Process` (Property 11):

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

```
( t <= Now
& t > Change [ 2 ]  ( Terminal_Mode )
& past ( Terminal_Mode, t )  = voter_mode )  )  )
```

which says, the mode of the terminal is set to *chirping* if there is no user input to the DRE within ten time units since last 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 action on the tape (Property 14). 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.

```
( 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
     & past ( Tape ( tapePosition ) , t )  =
        Make_Print_VoteEntry ( My_DRE.pickedName,
                 My_DRE.currentRace, My_DRE.pickedValue ) ) )
```

The above code states the vote entry (i.e., a record that consists of a candidate, race, and 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; we omit the start and summary conjuncts.

Specifying the integrity of the election results (Property 18.4): With this property, we want to guarantee that, after the election is closed, the results downloaded into the master PEB must be equal to the sum of the results collected from each DRE. The property is specified in the global invariant clause as

```
/*After election, downloaded results into the master PEB must be equal to
 the results produced by all DREs. */
  EXISTS p: PEB_Number
    ( the_PEB [ p ] .Kind = Master
    & FORALL d: DRE_Number
        ( the_PEB [ p ].ResultDownload_Completed ( the_DRE [ d ] )
        & the_DRE [ d ].Which_Phase = Post_Voting
        & the_DRE [ d ].DRE_State = Closed
        & FORALL C: Candidate, R: Race
             ( the_PEB [ p ].Candidates_Of_Race ( R ) =
                the_DRE [ d ].Race_Candidates ( R )
             & C ISIN the_DRE [ d ].Race_Candidates ( R )
             -> the_PEB [ p ].tabulatedData ( C, R, the_DRE [ d ] ) =
                 the_DRE [ d ].TotalTallyCount ( C, R ) ) ) )
&  /*Downloaded results in the master PEB must be equal to the
printed votes on the RTAL tape.*/
  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 ] )
        & 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 ] )
                = CountSelected ( C, R, the_RTAL [ rt ] )  -
                   CountCancelled ( 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`.

In the above code, the `CountSelected` and `CountCancelled` are definitions make the specification more readable. More specifically, they respectively introduce predicates which are used in our specification of the voting process to specify how many selections and cancelations have been printed for each candidate `C` who ran for race `R` in that particular RTAL printer `rt`.

This way, the requirements listed in Section 5.2.2 are converted into ASTRAL invariants, schedules, and constraints for each corresponding process instance. We need to be clear that we did not convert all the requirements to their ASTRAL equivalent in the way we describe them informally.

5.5.3 Formal Verification and Results

We used the ASTRAL and the PVS (Owre et al. 1993) analysis tool to analyze that the specification of the ES&S system meets the critical properties articulated previously. The main goal of our analysis is to provide the maximum assurance that the ES&S specification meets its critical requirements. To do that, it is necessary to generate proof obligations for critical requirements and prove them.

ASTRAL supports two kinds of proof obligations: correctness proofs

and consistency proofs. In the former case, the critical requirements of the system are proven to hold based on the executions of each process. In the latter case, whereas it is proven that any assumptions made in the system are never false. Generally, both proofs obligations are useful for the e-voting systems in order to run a correct and secure election. However, we particularly focus on the correctness proofs of the ES&S voting system. We mainly attempt to prove the invariants and schedules clauses related to the components (i.e., DRE, RTAL, PEB, and CF Card) in isolation and the system as whole to hold at all times.

5.5.3.1 Formal Verification

Before attempting the proof with a theorem prover, we should assure that the specification contains as few errors as possible, by performing a sequence of less costly steps.

Quoting from Kolano PhD thesis (Kolano 1999)

> *"Performing proofs within a mechanical theorem prover can take a significant amount of time and effort. Thus, there is a large overhead associated with finding errors in a specification. The more errors that are present in a specification when theorem proving begins, the more times a particular proof must be attempted before it can be completed. Thus, it is desirable to be as confident as possible that a specification is correct before a theorem prover is invoked."*

In ASTRAL, these steps include model checking and proof sketch construction. We applied the proof sketch construction strategies (such as proof ordering, transition steps, global and imported variable obligations, etc) for our purpose.

The specification of the ES&S system was first constructed and type-

checked using the ASTRAL SDE. Thereafter, we validated the specification and generated the corresponding proof obligations for the critical requirements. It is important to emphasize that the generated proof obligations are that of the intra-level proof obligations, which deal with proving that each process level satisfies its stated critical requirements and that the top level specification is consistent and satisfies the global requirements . Moreover, the specification was automatically translated into its PVS 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 and classify the ASTRAL specification into collections of simpler properties that infer the whole clause so that the proof of each property could be tackled separately. The splitter can be invoked on any section of an ASTRAL specification that resolves to a boolean expression. Table 5.1 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 ASTRAL SDE for which we discharged the proof commands.

5.5.3.2 Analysis Results

The assurance of the ES&S specification cannot be achieved without performing system proofs within the theorem prover. So far we managed to formally verify that the specification satisfies many of the critical requirements that we discussed previously, mostly the local invariants and constraints. The proofs were achieved by following the techniques presented in (Kolano 1999). 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 require-

	Proof Obligations	After Splitting
	Invar, Constr, Sched	Invar, Constr, Sched
DRE	4, 6,1	10, 9, 2
RTAL	1, 1, 3	1, 1, 3
PEB	1, 0, 1	2, 0, 1
CFCard	0, 0, 1	0, 0, 2
Global	6, 0, NA	9, 0, NA
Total	12, 7, 6	22, 10, 8
Total Proved		**13,7,3**

Table 5.1: Number of proof obligations and number of proved critical properties.

ments and applying the appropriate proof strategies developed to support the analysis of ASTRAL specifications. More specifically, we have proved 13 of the 22 invariants, 3 of the 8 schedules, and 7 of the 10 constraints (see Table 5.1). We expect that the other global and local properties can be proved using the same or similar proof techniques and strategies.

5.6 Extending the System Specification by Modeling Attack Scenarios

Analyzing a system in *non-nominal* situations —where some of its components are not behaving in the way they should be— has always been a challenge and, at the same time, interesting. Especially, this is common in systems engineering when performing safety assessment of critical systems. With this kind of assessment, usually an analyst specifies each individual behavior of a component of the system and extends it with its undesired behaviors in order to get what it is called extended model. Following that,

we analyze the resulting model that must satisfy certain behavioral properties that we would expect a correct system to satisfy.

We want to take a similar approach but we turn our attention to the analysis of the same security critical properties articulated previously against the extended model. The extended model is a combination of the original specification of the ES&S system which was discussed previously and attack scenarios which will shortly be discussed. More specifically, we extended the original specification with a set of transition specifications that represent known attacks that have been shown to successfully compromise the ES&S system. Each transition corresponds to a particular threat action for the voting system. Thereafter, we process the extended specification and automatically generate proof obligations related to the security requirements for the PVS analysis tool using the same strategy discussed in Section 5.5.3

In particular, we wish to prove the security properties against the extended model for the following interests:

- If all of the proof obligations were to be proved, then the system specification must be missing some critical security requirements, since the modeled attacks were already demonstrated to be successful. Therefore, it would be necessary to see what additional critical requirements are needed to disallow the threat actions and keep the extended specification from being proved.

- In contrast, not being able to prove the extended specification would indicate that one, or more, threat action violates at least one critical security requirement. However, since we know that attacks composed of these threat actions have been used to successfully compromise the system, it also indicates that there could be an implementation or specification error or an unsatisfied procedural assumption that results

in the actual system or the environment not satisfying their respective formal specification.

5.6.1 Attack Specifications

We model the attack scenarios presented in Section 5.2.3 in terms of threat actions expressed as ASTRAL transition specifications. The system model is extended by augmenting the specification with new possible states that are the result of the execution of the threat actions.

In particular, the attack scenarios are encoded to extend the original specification of the ES&S voting system using the following strategies:

1. we define new types, variables, and constants. There are two kinds of variables that we declare: those that provide additional information about the state of the system (e.g., the system is now about to display the review ballot) and those that hold information about the successful execution of a threat action (e.g., a fleeing voter has been faked).

2. a transition is defined for each threat action, which is part of a given attack scenario. Note that one attack scenario can be implemented using one or more threat actions.

3. a transition may be split into two or more transitions, or a transition may be extended with more information to specify the attack scenario.

We assume that the attacker can intercept the normal voting process at any point. For instance, if s/he intercepts the process before the review screen is displayed and the attack is successful, then the `tempVoteRecord` variable should include the maliciously modified candidate and the `Display` variable should update the screen accordingly. It is, in fact, the voter's task to correctly verify that what is displayed exactly matches her/his preferences.

To represent the various kinds of voters (unattentive, careful, and fleeing), we introduced the following global type

```
TYPE
   VoterType : (unattentive, careful, fleeing)
```

In addition, the variables

```
VARIABLE
   vote_changed, Fleeing_Faked: Boolean,
   attPickedName : Name
```

are declared to, respectively, hold information about whether the voter's vote is changed (obviously, by a successful attack action), whether the fleeing voter is faked, and the name of the attacker's candidate. In addition, information about where the attacker intercepts the process to start the threat action is encoded by the (boolean) variables `review_displayed` and `summary_sent2RTAL`. Namely, they respectively hold information about the attack action happened just before reviewing the final votes and right after the summary data are sent to the printer.

When an attacker changes or cancels a vote, it is actually performing a sequence of interactions with the DRE process in order to fulfill the threat action. The successful completion of such an action eventually assigns new values to some of the exported variables above discussed.

The following transition specifies the *change vote threat action*, which appears in the sequence diagram depicted in Figure 5.2.

```
TRANSITION Attack_Change_Vote(vc, ac :Candidate, vType: VoterType)
  ENTRY       [TIME ACV_Dur]
     Which_Phase = During_Voting & Terminal_Mode = voter_mode
     & vType = Unattentive
     & EXISTS R: Race (
          vc ISIN Displayed_Candidates ( R )
          & ac ISIN Displayed_Candidates ( R )
```

```
          & vc ISIN tempVoteRecord ( R )
          & Picked (Candidate_Name ( vc ), Race_Title ( R ) ) )
          & ~Picked ( Candidate_Name (ac), Race_Title ( R ) ) )
    & vc ~= ac
    & EXISTS b: Button ( b = REVIEW & Button_Pushed ( b )  )
    & scrName = REVIEW_SCREEN & ~Review_Displayed & ~Vote_Changed
EXIT
    EXISTS R: Race (
        vc ISIN Displayed_Candidates' ( R )
        & ac ISIN Displayed_Candidates' ( R )
        & vc ISIN tempVoteRecord' ( R )
        & tempVoteRecord ( R )  BECOMES
          (tempVoteRecord' ( R ) SET_DIFF {vc} ) UNION  { ac }
        & Picked ( Candidate_Name ( vc ), Race_Title ( R ) ) = FALSE
        & Picked ( Candidate_Name ( ac ), Race_Title ( R ) ) = TRUE
        & FORALL CN:Name, R1:Title (
          ( (  CN ~= Candidate_Name ( vc )
            & CN ~= Candidate_Name ( ac ) )
           | R1 ~= Race_Title (R ) )
         -> Picked (CN, R1) = Picked' (CN,R1) ) & currentRace = R )
  & Signal_Enabled & Which_Signal = Vote_Signal
  & pickedName = Candidate_Name ( vc )
  & attPickedName = Candidate_Name ( ac ) & Vote_Changed
```

The enabling condition for this threat action (i.e., for the transition) specifies that the fleeing voter is voting during election period in the voter's terminal mode, that there exists a race R such that the voter's candidate vc is in the displayed candidates list for race R for which the voter already voted, that the attacker's candidate ac is also a legitimate candidate contained in the displayed list for the same race R and it is not selected by the voter, and the voter desire is different from the attacker (i.e., vc $\sim$ = ac). In addition the voter has already requested the review screen, currently there is nothing shown on the REVIEW_SCREEN, and there is no change of

vote at the moment.

After the threat action is successfully executed (i.e., after the transition is ended) the following holds: the voter's selection contained in the `tempVoteRecord` now contains the attacker's choice, the `Picked` value is *true* for `ac` and is `FALSE` for voter candidate `vc`. In addition, the exported variables `currentRace`, `pickedName`, `attPickedName`, `pickedValue` and `Which_Signal` have new values, and the signaling variable is *true*. This indicates that the RTAL can now print the modification expressed in these exported variables. The RTAL process prints this information by executing the `Print_Selection` transition (the details of this transition can be found in the Appendix A).

The above modification, which is contained in the `tempVoteRecord` variable, is also displayed on the review screen. It is worth noting that both the review screen and what is printed on the RTAL tape report the modified selection, rather than the original one. From an attacker's point of view, it is better to keep the display and tape consistent, because, if an abnormality is detected, then it is more likely to be attributed to a display miscalibration rather than to an attack. It is possible that the voter will detect such a change. In this case, the voter can recast his/her vote by calling the `Push_Button` and `Make_Selection` transitions.

The *Faking a Fleeing Voter* attack is an example of a scenario that requires several threat actions. The *canceling of votes* by faking a fleeing voter has three threat actions (as depicted in the sequence diagram, see Figure 5.3). The threat actions are specified as three transitions in the DRE process:

1. *Attack_Change_Vote.* This is an except/exit transition that specifies the fact that an attacker fakes a fleeing voter by pretending to complete the voting process on her/his behalf. The exit assertion of this transition will set the variable `Fleeing_Faked` to *true*.

2. *Attack_ReDisplay.* This transition specifies the fact that after some delay (during which time the voter leaves the booth) since the voter is successfully fooled, the attacker directs the DRE to display the confirmation page again.

3. *Attack_Call_ChirpingR.* This specifies that after the voter has been fooled and *DelayTime* has passed, the attacker resumes the normal voting process by calling the chirping routine. This results in the poll worker taking action according to the prescribed procedure. (This transition is only enabled after the first two transitions have been executed.)

We say the *canceling of votes* is successful only after transition #3 has been executed. We omit the formal specifications for these threat actions, which are specified similarly to the others described earlier.

5.6.1.1 Verifying the Extended Specification

After extending the system specification with the threat actions, the ASTRAL SDE is used to generate the proof obligations for the extended specification. Because there are additional transitions, there are more proofs to be done. In addition, because some of the original transitions were split and/or extended, the corresponding proof obligations must be reproved. In order to prove, the requirement we followed the same procedure discussed in Section 5.5.3. However, the proofing process is very complex in the extended model.

We have just begun the verification process for the extended specification. We started with the proof obligations that were unchanged to assure that they are still valid. So far, we have reproved 4 of the 13 invariants and 2 of the 7 constraints (see Table 5.2).

	Proof Obligations	After Splitting
	Invar, Constr, Sched	Invar, Constr, Sched
DRE	4, 6,1	10, 9, 2
RTAL	1, 1, 3	1, 1, 3
PEB	1, 0, 1	2, 0, 1
CFCard	0, 0, 1	0, 0, 2
Global	6, 0, NA	9, 0, NA
Total	12, 7, 6	22, 10, 8
Total Proved		**4/13,2/7,0/3**

Table 5.2: Number of reproved proof obligations after extending the original specification with attack information.

5.7 Summary

In this chapter, we have shown how formal verification techniques can be used to model and reason about the security of e-voting systems. Our approach consists of formulating each individual component of the voting system as process instance in ASTRAL, specification and verification of critical security properties about individual component and the system as whole. On the other hand, we specified the attack scenarios that are reported by academic studies on the security testing and analysis of such systems. Namely, along the line of nominal behavior specification, we model and specify attacks. We extend the specification that describe the nominal behaviors of the system under analysis by augmenting the attack model. Thereafter, we attempted to analyze the extended model against the same set of critical requirements as the nominal model should meet. The threat actions that we specified in the extended system were those needed to model the specific scenarios presented in (McDaniel et al. 2007).

We admit that the proof results we presented are not complete. The reason is, when we started proving the proof obligations for the original system specification we were 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 plan to have all of the properties proved in the near future.

Although our approach provides certain benefits over existing works in the area, it is in no way a verification "silver bullet". As with any formal verification technique, it requires the use of formal languages, various analytic tools including a theorem-proving system, and considerable skill on understanding the various information sources as well as understanding various components of the system. Additionally, we must be clear that this research work does not consist of a novel specification technique nor a novel voting system. However, it clearly shows how formal methods can be effectively used for the specification and verification of e-voting systems in order to guarantee the correctness of the system.

Chapter 6

Conclusion

6.1 Summary and Discussion

The fast-paced diffusion of ICTs in numerous walks of life (e.g., public administration) not only offers attractive benefits but also emerges with side-issues worth considering. Deploying ICTs in a safe and secure manner requires ensuring the technical and procedural levels of assurance with respect to social and regulatory frameworks. However, existing remedies are not matured enough to embrace procedural implications and the need for multidisciplinary approach on the safe and secure operation of system.

The usage of techniques and tools (such as BPR, software engineering and formal methods) for modeling, specifying, analyzing, as well as developing a system has been receiving much effort from several active research groups. However, there are still many open issues for research, e.g., a method that considers procedures as part of the analysis is absent. Note that threats and attacks may not only derive from pitfalls in the systems, but also from ill-designed procedures. Under this view, one must be cognizant of how endowing a system with a violation of its procedures impacts the system's goals. These challenges, among others, motivated this work.

In spite of the potential advantages e-voting brings 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 cause of 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 the poor design and implementation of (some of) the systems currently deployed for elections in the US and other countries, as different studies have reported and demonstrated. Such weaknesses expose e-voting systems, and consequently elections, to threats and attacks, resulting in effects ranging 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). As a consequence, this would contribute for the failure to achieve public confidence or to meet the highest democratic standards.

The main contributions of this work are: (i) a methodology for process modeling and the provision of tool support within a general-purpose tool offering easy-to-understand graphical description techniques. The notations and approach basically conform to the approaches proposed in the past, and the added value is a set of rules and conventions that simplify maintenance; (ii) formal methods based approach for security assessment of a system and procedures; and (iii) the demonstration of the practical applicability of the presented methodologies and the tool in socially relevant domain, i.e., in electoral domain.

Process Modeling. The presented tool-supported methodology takes BPR concepts as the core elements for business process modeling and analysis in favor of PA processes. It provides a systematic modeling guidance to support functional analysts in PA. The approach, on one hand, is used to obtain unambiguous, standard (with respect to the requirements listed in 3.1.2), and objective process models from two views. These are static and dynamic views, using the facilities of UML notations respectively use

cases and activity diagrams. On the other hand, it helps to link the models with their legal framework (in our case: the laws from which the models are constructed), increasing the traceability between them. With this, various users who involve in development activity are benefited. E.g, law makers elaborate models in collaboration with software developers or process engineers, and understand the impact of law or process changes as a result of interventions due to people, processes, and technology. The VLPM tool is employed to support the methodology with a systematic extraction and generation of process models (as "process-tree") from their XML representation. The applicability of the methodology and the tool has been tested within the ProVotE project. Notice that, even if we applied the approach mainly in electoral domain, the methodology and the tool presented in this work can be used in other domains with some customization effort, if applicable. For that matter, we tested their applicability by taking example from Italian Immigration law, see in (Ciaghi et al. 2009b).

One significant limitation of the tool is that it does not provide notations and means to represent the principles behind the procedures and to reason about possible alternative implementation. Even from the business re-engineering point of view, such principles represent an essential part, since they provide the framework and the constraints for the definition of new procedures and laws. This, in turn, "moves" part of the re-engineering activity back to the "natural language" domain, where inconsistencies and ambiguities might arise. Moreover, the current implementation of the tool is based on the Italian law representation system.

Security Analysis. The proposed approach is aimed to tackle (some of) the challenges in security analysis of processes, which we called procedural security analysis. Our work builds upon and improves those presented in the state of the art by widening the scope of the analysis and by taking into account aspects related to threats, procedures, and interaction of the sys-

tem with its environment. Namely, we developed a generic assets-centered methodology for the analysis. Using the approach, we have showed how to encode executable models describing the nominal procedures using NuSMV input language —mainly from the dynamic view of process models— and to extend such models with possible attacks, by assuming all possible combination of attacks can be made at each execution step. We have tested the applicability of the proposed approach using core use cases taken from the ProVotE e-voting system. Among the advantages of the approach the possibility of reasoning about threat composition (e.g., coordinated attacks; complex and unforeseen attacks resulting from the composition of elementary threats), and the possibility of reasoning about evolution of assets over time.

More importantly, the benefits that the proposed approach offers are the following. First of all, using the methodology and the machinery of NuSMV tool, it is possible to understand what are the hypotheses and conditions under which a given security goal is achieved or breached. One can also use the results of the analyzes to devise requirements that make organizational and (software) systems more secure. Finally, we believe that, such results are foundations to familiarize actors (election officials or polling officers) with the procedural threats and attacks that happen during elections, or otherwise anytime during the electoral phases, by complementing works discussed in the state of the art.

Reverse Synthesis We have showed how formal verification techniques can be used to model and reason about the security of e-voting system. The methodology proposed for procedural security analysis is customized and applied in the reverse synthesis. We believe that besides analyzing the system against its requirements, it is equally important to perform an analysis under malicious circumstances where the system execution model is enriched with attack behavior. This is helpful in order to detect miss-

ing requirements or unwarranted assumptions about the specification we developed. In addition, this allows to sketch counter-measure strategies to be used when the system behaves differently than it should and to build confidence about the system under development.

Since formal methods have been recognized as powerful and effective approaches for improving the security and quality of complex systems (such as in flight (software) system), drawing straight connection with this can help making better the current development of e-voting machines. There exist several works that have been done for a system level analysis and testing of existing e-voting systems, mainly providing a low level assurance. Although several points remain to be addressed, this work is an important step towards a substantially higher assurance attempting to bridge the missing gap.

The success of the next generation of e-voting machines depends upon being able to capitalize from the lessons learned by using and analyzing the systems currently deployed. The work we have presented in this dissertation is one way in which we can get a better understanding of the strengths and the weaknesses of existing systems and thus lay the foundations for engineering and deploying a new generation of more secure and robust technologies for polling stations.

6.2 Future Work

Currently, follow-up work and further evaluation of the approach in this work are on going. In line with the stated limitations, first we will evaluate the VLPM tool extension to customize XML standard to support schema adopted by other nations and allow for the integration of UML tools providing standard connection "ports" to the UML (e.g., XMI). Second, we will work on further integration of the VLPM approach with goal-based frame-

work named *Nomos*, as it offers alternative reasoning at higher level. For that matter, a very preliminary proposal is published in (Villafiorita et al. 2010). Finally, it would be interesting to support the approach with formal methods. This means that adding new functionalities say, e.g., "ports" to FSAP/NuSMV-SA (see the next paragraph also), allowing automatic generation of specification into target language (e.g., NuSMV input language), so that formal verification can be supported.

In the procedural security analysis, some research issues remain open. In our future work, we will consider multiple instance of an asset in the analysis. The threat-actions considered in the cast study should be enriched with more threat-actions. In fact, the consideration of the threat-actions depends on the case study we choose, thus we need to precisely isolate generic threat-actions, such that they can be customized per the case study chosen, if applicable. Furthermore, we are currently investigating the possibility of automating the threat injection on top of the FSAP/NuSMV-SA platform. Specifically, we plan to develop a tool in order to integrate and/or extend the current usage scenario of the platform, with emphasis on procedural security analysis. The definition of a set of generic library of attack models corresponding to threat-actions is part of the future work. Additionally, we will conduct further research on the possibilities of integrating our approach in the Common Criteria (Common Criteria 2007) methodology and on the evaluation procedures discussed in (Volkamer and McGaley 2007, Volkamer 2009).

With respect to the application of formal methods, future work moves along the line of the two main lessons drawn from reverse synthesis approach. Based on the current results, we would like to provide a *generic* specification for DRE-based e-voting machines. This generic specification could then be used as a basis for the specification and design of a new generation of e-voting systems. Additionally, the threat actions that we have

specified in the extended system were those needed to model the specific scenarios presented in (McDaniel et al. 2007). They are a minimal sampling of the possible threat actions, but they demonstrate the approach on a real system. In the future we would like to model more general threat actions to see if new attack scenarios or missing critical requirements can be identified.

Bibliography

Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *SIGPLAN Not.*, 36(3):104–115, 2001. 40

Ben Adida. *Advances in Cryptographic Voting Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August 2006. 8, 36, 37

Paul Alpar and Sebastian Olbrich. Legal Requirements and Modelling of Processes in e-Government. *Electronic Journal of e-Government*, 3, 2005. 19, 50, 52

Rachid Anane, Richard Freeland, and Georgios K. Theodoropoulos. e-Voting Requirements and Implementation. *E-Commerce Technology, IEEE International Conference on, and Enterprise Computing, E-Commerce, and E-Services, IEEE International Conference on*, 0:382–392, 2007. 2

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. 10, 39

A. Antoniou, C. Korakas, C. Manolopoulos, A. Panagiotaki, D. Sofotassios, Paul G. Spirakis, and Yannis C. Stamatiou. A Trust-Centered Approach

for Building E-Voting Systems. In Maria Wimmer and Hans Jochen Scholl and Åke Grönlund, editor, *EGOV*, volume 4656 of *Lecture Notes in Computer Science*, pages 366–377. Springer, 2007. 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. ISSN 1545-5971. 1, 4

D. Balzarotti, G. Banks, M. Cova, V. Felmetsger, R. Kemmerer, W. Robertson, F. Valeur, , and G. Vigna. An Experience in Testing the Security of Real-world Electronic Voting Systems. *IEEE Transactions on Software Engineering*, 2010. 39

Davide Balzarotti, Greg Banks, Marco Cova, Viktoria Felmetsger, Richard Kemmerer, William Robertson, Fredrik Valeur, and Giovanni Vigna. Are Your Votes Really Counted?: Testing the Security of Real-world Electronic Voting Systems. In *ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis*, pages 237–248, New York, NY, USA, 2008. ACM. 3, 10, 39, 101

David A. Basin, Jürgen Doser, and Torsten Lodderstedt. Model Driven Security for Process-Oriented Systems. In *SACMAT*, pages 100–109, 2003. 28

Josh Benaloh and Dwight Tuinstra. Receipt-Free Secret-Ballot Elections (extended abstract). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 544–553, New York, NY, USA, 1994. ACM. 37

Kamal Bhattacharya, Cagdas Evren Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards Formal Analysis of Artifact-Centric Business Pro-

cess Models. In Gustavo Alonso and Peter Dadam and Michael Rosemann, editor, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2007. 18, 111, 114

Andrzej Bialas. Information Security Systems vs. Critical Information Infrastructure Protection Systems - Similarities and Differences. In *Proceedings of the International Conference on Dependability of Computer Systems*, pages 60–67, Washington, DC, USA, 2006. IEEE Computer Society. 25

Andrzej Bialas. Semiformal Approach to the IT Security Development. In *Proceedings of the 2nd International Conference on Dependability of Computer Systems*, pages 3–10, Washington, DC, USA, 2007. IEEE Computer Society. 25

Matt Bishop. *Computer Security Art and Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. 1, 100

Matt Bishop and David Wagner. Risks of e-voting. *Commun. ACM*, 50 (11):120–120, 2007. 2, 3

Marco Bozzano and Adolfo Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *Int. J. Softw. Tools Technol. Transf.*, 9(1):5–24, 2007. 93, 95, 141

Sviatoslav Braynov and Murtuza Jadiwala. Representation and Analysis of Coordinated Attacks. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 43–51, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-781-8. 21, 27, 108

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. 3

Volha Bryl, Fabiano Dalpiaz, Roberta Ferrario, Andrea Mattioli, and Adolfo Villafiorita. Evaluating Procedural Alternatives: a Case Study in e-voting. *EG*, 6(2):213–231, 2009. 33

California Secretary of State. Withdrawal of approval of diebold election systems, inc., gems 1.18.24/accuvote-tswaccuvote-os dre & optical scan voting system and conditional re-approval of use of diebold election systems, inc., gems 1.18.24/accuvote-tsx/accuvote-os dre & optical scan voting system. http://www.sos.ca.gov/elections/voting_systems/ttbr/diebold_102507.pdf, October 2007. 144

Stefano Campanelli, Alessandro Falleni, Fabio Martinelli, Marinella Petrocchi, and Anna Vaccarelli. Mobile Implementation and Formal Verification of an e-Voting System. In *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, Washington, DC, USA, 2008. IEEE Computer Society. 4, 40

Letizia Caporusso, Carlo Buzzi, Giolo Fele, Pierangelo Peri, and Francesca Sartori. Transition to Electronic Voting and Citizen Participation. In Robert Krimmer, editor, *Electronic Voting*, volume 86, pages 191–200. GI, 2006. 41, 82

Richard T. Carback, Stefan Popoveniuc, Alan T. Sherman, , and David Chaum. Punchscan with Independent Ballot Sheets: Simplifying Ballot Printing and Distribution with Independently Selected Ballot Halves. In *In Preproceedings of the 2007 IAVoSS Workshop on Trustworthy Elections (WOTE 2007)*, 2007. 36

Nuno Castela, Jose M. Tribolet, Alberto Silva, and Arminda Guerra. Busi-

ness Process Modeling with UML. In *ICEIS (2)*, pages 679–685, 2001. `citeseer.ist.psu.edu/article/castela01business.html`. 15

Caterina Lupo. NormeinRete: a federative approach to on-line legislation access. In *International Conference Parliaments' Information Management in Africa: Challenges and Opportunities of ICTs to Strengthen Democracy and Parliamentary Governance, Nairobi, Kenya,*, February 2005. 56, 57

D. Chaum, R.T. Carback, J. Clark, A. Essex, S. Popoveniuc, R.L. Rivest, P. Ryan, E. Shen, A.T. Sherman, and P.L Vora. Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, 2009. 37

David Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security and Privacy*, 2:38–47, 2004. 36

David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi L. Vora. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security & Privacy*, 6(3):40–46, 2008. 37

Aaron Ciaghi, Andrea Mattioli, and Adolfo Villafiorita. Vlpm: a tool to support bpr in public administration. In *Proceedings of the Third International Conference on Digital Society (ICDS2009)*, pages 289–293. IEEE Computer Society, 2009a. 55, 64

Aaron Ciaghi, Adolfo Villafiorita, Komminist Weldemariam, Andrea Mattioli, and Quoc-Sang Phan. Supporting public administration with an integrated bpr environment. In *AFRICOMM 2009,*. ICST, 2009b. 88, 205

Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An Open Source Tool for Symbolic Model Checking. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 241–268. Springer Berlin / Heidelberg, January 2002. 6, 118

Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, 2000. 141

Kevin J. Coleman, Joseph E. Cantor, and Thomas H. Neale. Crs report for congress: Presidential elections in the united states: A primer. http://www.senate.gov/reference/resources/pdf/RL30527.pdf, April 17 2000. 30

Common Criteria. Common Criteria for Information Technology Security Evaluation, 2007. `http://www.commoncriteriaportal.org/`. 1, 20, 23, 208

Council of Europe. *Recommendation on legal, operational and technical standards for e-voting*. Council of Europe, September 2004. 34

Ronald J.F. Cramer, Matthew Franklin, L. A.M. Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. Technical report, CWI (Centre for Mathematics and Computer Science), 1995. 36

Kristopher Daley, Ryan Larson, and Jerad Dawkins. A Structural Framework for Modeling Multi-Stage Network Attacks. In *ICPPW '02: Proceedings of the 2002 International Conference on Parallel Processing Workshops*, page 5, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1680-7. 22

Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993. 22

Thomas H. Davenport. *Process innovation: reengineering work through information technology.* Harvard Business School Press, Boston, MA, USA, 1993. 14

Thomas H. Davenport and James E. Short. The New Industrial Engineering: Information Technology and Business Process Redesign. *Sloan Management Review*, 31(4):11–27, 1990. 14

João Alberto de Oliveira Lima, Monica Palmirani, and Fabio Vitali. 'http' or 'urn' URIs for legal resources? how about both? In *Jurix 2007 Workshop on Legislative XML*, 2007. 57

Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17(4):435–487, 2009. 4, 40

Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic Verification of Data-Centric Business Processes. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 252–267, New York, NY, USA, 2009. ACM. 18

Dominique Bolignano. Formal Methods and Security Evaluation (Invited Talk). In *TPHOLs '99: Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, pages 291–292, London, UK, 1999. Springer-Verlag. 25

Marlon Dumas and Arthur H. M. ter Hofstede. Uml activity diagrams as a workflow specification language. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Lan-*

guages, Concepts, and Tools, pages 76–90, London, UK, 2001a. Springer-Verlag. 15

Marlon Dumas and Arthur H. M. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 76–90, London, UK, 2001b. Springer-Verlag. ISBN 3-540-42667-1. 15

Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997. 33

Golnaz Elahi and Eric S. K. Yu. A goal oriented approach for modeling and analyzing security trade-offs. In Christine Parent and Klaus-Dieter Schewe and Veda C. Storey and Bernhard Thalheim, editor, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2007. 22

Rik Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, Centre for Telematics and Information Technology (CTIT) University of Twente, P.O. Box 217, 7500 AE Enschede The Netherlands, 2002. 15, 17

Rik Eshuis. Symbolic Model Checking of UML Activity Diagrams. *ACM Trans. Softw. Eng. Methodol.*, 15(1):1–38, 2006. 17

Rik Eshuis and Roel Wieringa. Tool Support for Verifying UML Activity Diagrams. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 30(7), 2004. 18

Aleks Essex, Jeremy Clark, Richard Carback, and Stefan Popoveniuc. Punchscan in practice: An E2E election case study. In *In Preproceedings*

of the 2007 IAVoSS Workshop on Trustworthy Elections (WOTE 2007), 2007. 36

David Evans and Nathanael Paul. Election Security: Perception and Reality. *IEEE Security and Privacy Magazine*, 2:24–31, 2004. 31

Federal Election Commission. *2002 Voting System Standards*. United States Election Assistance Commission, Available from http://www.eac.gov/, 2002. 34

Federal Election Commission. *2005 Voluntary Voting System Guidelines (VVSG)*. United States Election Assistance Commission, http://www.eac.gov/, 2005. 34

Igor Nai Fovino and Marcelo Masera. Through the Description of Attacks: A Multidimensional View. In *SAFECOMP 2006*, Lecture Notes in Computer Science, pages 15–28. Springer-Verlag, 2006. 21, 93

Rune Fredriksen, Monica Kristiansen, Bjørn Axel Gran, Ketil Stølen, Tom Arthur Opperud, and Theodosis Dimitrakos. The CORAS Framework for a Model-Based Risk Management Process. In *SAFECOMP '02*, pages 94–105, London, UK, 2002. Springer-Verlag. 4, 25

Christian Fritz, Richard Hull, and Jianwen Su. Automatic Construction of Simple Artifact-Based Business Processes. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 225–238, New York, NY, USA, 2009. ACM. 18

Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251, London, UK, 1993. Springer-Verlag. ISBN 3-540-57220-1. 36, 37

Ryan Gardner, Sujata Garera, and Aviel Rubin. On the Difficulty of Validating Voting Machine Software with Software. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology on USENIX/Accurate Electronic Voting Technology Workshop*, pages 11–11, Berkeley, CA, USA, 2007. USENIX Association. 3, 39

Cagdas E. Gerede and Jianwen Su. Specification and Verification of Artifact Behaviors in Business Process Models. In Bernd J. Krämer and Kwei-Jay Lin and Priya Narasimhan, editor, *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2007. 18, 111, 114

Cagdas E. Gerede, Kamal Bhattacharya, and Jianwen Su. Static Analysis of Business Artifact-centric Operational Models. In *SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, pages 133–140, Washington, DC, USA, 2007. IEEE Computer Society. 18

Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus, 2000. 17

Jaap Gordijn, Hans Akkermans, and Hans van Vliet. Business modelling is not process modelling. In *Conceptual Modeling for E-Business and the Web*, Lecture Notes in Computer Science, pages 40–51. Springer Berlin / Heidelberg, 2000. 14

Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Mining Hierarchies of Models: From Abstract Views to Concrete Specifications. In *Business Process Management*, pages 32–47, 2005. 19

Dimitris A. Gritzalis. *Secure Electronic Voting*. Kluwer Academic Publishers, 2003. 2, 31

Anthony Hall. Seven Myths of Formal Methods. *IEEE Trans. Softw.*, 7, 1990. 4

Ferrel Heady. *Public administration : a Comparative Perspective*. Public administration and public policy ; 41. Fourth edition edition, 1991. 48

Constance L. Heitmeyer. On the role of formal methods in software certification: An experience report. *Electronic Notes in Theoretical Computer Science*, 2009. 25

Constance L. Heitmeyer, Myla Archer, Elizabeth I. Leonard, and John McLean. Applying Formal Methods to a Certifiably Secure Software System. *IEEE Trans. Software Eng.*, 34(1):82–98, 2008. 4

Vlatka Hlupic. Business Process Modelling Using Discrete Event Simulation: Potential Benefits And Obstacles For Wider Use,. *International Journal of Simulation: Systems, Science and Technology*, 7:62–67, 2003. 15

Arthur H. M.ter Hofstede and Maria E. Orlowska. On the Complexity of Some Verification Problems in Process Control Specifications. *Comput. J.*, 42(5):349–359, 1999. 16

Ida Hogganvik. *A Graphical Approach to Security Risk Analysis*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo., 2007. 25, 93

Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003. 16

Daisuke Horie, Shoichi Morimoto, and Jingde Cheng. A Web User Interface of the Security Requirement Management Database Based on ISO/IEC 15408. In *International Conference on Computational Science (4)*, Lecture Notes in Computer Science, pages 797–804. Springer, 2006. 25

Pao-Ann Hsiung, Yean-Ru Chen, and Yen-Hung Lin. Model Checking Safety-Critical Systems Using Safecharts. *IEEE Trans. Comput.*, 56(5): 692–705, 2007. 95

Richard Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges,. In *OTM '08: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems,*, pages 1152–1163, Berlin, Heidelberg, 2008. Springer-Verlag. 111

Inc. ES&S. Election Systems & Software: iVotronic TM Voting System. Version 9.1.x Election Day Operations Checklist, Revision Date: January 2007. 7, 150, 159

Kenneth R. Iversen. A Cryptographic Scheme for Computerized Elections. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 405–419, London, UK, 1991. Springer-Verlag. 36

Jan Jürjens. Model-based security engineering. In *SECRYPT*. INSTICC Press, 2006. 28

Jintae Lee. Goal-based process analysis: a method for systematic process redesign. In *Proceedings of the Conference on Organizational Computing Systems, COOCS 199*, pages 196–201. ACM, 1993. 15

Andy Jones and Debi Ashenden. *Risk Management for Computer Security: Protecting Your Network & Information Assets*. Butterworth-Heinemann, Newton, MA, USA, 2005. 2

Douglas W. Jones. *The Evaluation of Voting Technology*, chapter 1, pages 3–16. Advances in Information Security. Kluwer Academic, 2003. 39

Jin-Young Choi Junkil Park. Formal security policy model for a common criteria evaluation. In *ICACT2007*. IEEE, 2007. 25

Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006. ISBN 1904811817. 15

Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, London, UK, 2002. Springer-Verlag. 28

Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: a systems perspective. In *Proceedings of the 14th conference on USENIX Security Symposium*, 2005. 36, 37

Richard A. Kemmerer. Integrating Formal Methods into the Development Process. *IEEE Software*, 7(5):37–50, 1990. 4

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. doi: {http://doi.ieeecomputersociety.org/10.1109/SECPRI.2004.1301313}. 3, 10, 39, 101

Paul Z. Kolano, Zhe Dang, and Richard A. Kemmerer. The Design and Analysis of Real-Time Systems Using the ASTRAL Software Development Environment. *Ann. Softw. Eng.*, 7(1-4):177–210, 1999. 7, 161

P.Z. Kolano. *Tools and Techniques for the Design and Systematic Analysis of Real-Time Systems*. PhD thesis, University of California, Santa Barbara, 1999. 160, 191, 192

Adolfo Villafiorita Komminist Weldemariam and Andrea Mattioli. Experiments and Data Analysis of Electronic Voting System. In *CRiSIS '09:*

Forth International Conference on Risks and Security of Internet and Systems, pages 249–254. IEEE Computer Society, October 2009. 11, 41

M. Koubarakis and D. Plexousakis. Business Process Modelling and Design — A Formal Model and Methodology. *BT Technology Journal*, 17(4): 23–35, 1999. 16

Manolis Koubarakis and Dimitris Plexousakis. A Formal Model for Business Process Modeling and Design. In Benkt Wangler and Lars Bergman, editor, *CAiSE*, Lecture Notes in Computer Science, pages 142–156. Springer, 2000. 16, 18, 108, 114

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)*, Lecture Notes in Computer Science, pages 186–200, Edinburgh, U.K., April 2005. Springer. 4, 40

Vitus S.W. Lam and Julian A. Padget. Symbolic Model Checking of UML Statechart Diagrams with an Integrated Approach. In *ECBS '04: Proceedings of the 11th IEEE International Conference and Workshop on Engineering of Computer-Based Systems*, pages 337–347. IEEE Computer Society, 2004. 120

Costas Lambrinoudakis, Spyros Kokolakis, Maria Karyda, Vasilis Tsoumas, Dimitris Gritzalis, and Sokratis Katsikas. Electronic Voting Systems: Security Implications of the Administrative Workflow. In *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, page 467, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1993-8. 3, 33

Lehman, M. M. Process Modelling—where next. In *ICSE '97: Proceedings*

of the 19th international conference on Software engineering, pages 549–552, New York, NY, USA, 1997. ACM. 15

Richard Lenz and Manfred Reichert. IT Support for Healthcare Processes. In *Business Process Management*, pages 354–363, 2005. 19

Hector J. Levesque, Fiora Pirri, and Raymond Reiter. Foundations for the Situation Calculus. *Electron. Trans. Artif. Intell.*, 2:159–178, 1998. 17

Torsten Lodderstedt, David A. Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In Jean-Marc Jézéquel, Heinrich Hußmann, and Stephen Cook, editors, *UML*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2002. ISBN 3-540-44254-5. 28

Pericles Loucopoulos and Evangelia Kavakli. Enterprise Modelling and the Teleological Approach to Requirements Engineering. *Int. J. Cooperative Inf. Syst.*, 4(1):45–79, 1995. 16

Michael Lowry and Daniel Dvorak. Analytic Verification of Flight Software. *IEEE Intelligent Systems*, 13(5):45–49, 1998. 4

Caterina Lupo. Format for the electronic representation of regulatory measures through the XML markup language (In Italain). *Allegato tecnico alla Circolare 22 aprile 2002, n. AIPA/CR/40*, 2002. 56

Shuailiang Ma, Li Zhang, and Jimei He. Towards Formalization and Verification of Unified Business Process Model Based on Pi Calculus. In *SERA*, pages 93–101. IEEE Computer Society, 2008. 16

Yogesh Malhotra. Business process redesign: An overview. *IEEE Engineering Management Review*, 26(3), 1998. 14

Ragavan Manian, Joanne Dugan Bechta, David Coppit, and Kevin J Sullivan. Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems. In *HASE '98: The 3rd IEEE International Symposium on High-Assurance Systems Engineering*, pages 21–28, Washington, DC, USA, 1998. IEEE Computer Society. 95

Andrea Marchetti, Fabrizio Megale, Enrico Seta, and Fabio Vitali. Using xml as a means to access legislative documents: Italian and foreign experiences. *SIGAPP Appl. Comput. Rev.*, 1:54–62, 2002. 56

Fabio Martinelli. Symbolic Semantics and Analysis for Crypto-CCS with (Almost) Generic Inference Systems. In *MFCS '02: Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, pages 519–531, London, UK, 2002. Springer-Verlag. 40

Andrea Mattioli. Analisi dei Processi in Ambito di Voto Elettronico per le Elezioni in Provincia di Trento. Master's thesis, University of Trento, 2006. 10, 51, 55, 64

Sjouke Mauw, Radu Mateescu, and Wil Janssen. Verifying business processes using spin. In *Proceedings of the 4th. International SPIN Workshop*, pages 21–36, 1998. 16

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. 7, 39, 159, 200, 209

Margaret McGaley. *E-voting: an Immature Technology in a Critical Context.* PhD thesis, Departement of Computer Science, National University of Irelan, Maynooth, September 2008. 29, 34, 35, 37, 90, 151, 160

Kenneth L McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. 118

Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. A Common Criteria Based Security Requirements Engineering Process for the Development of Secure Information Systems. *Comput. Stand. Interfaces*, 29(2):244–253, 2007. ISSN 0920-5489. 24

Rebecca T. Mercuri. *Electronic Vote Tabulation Checks and Balances*. PhD thesis, University of Pennsylvania, 2001. 3, 29, 31, 32, 34, 35, 160

Rebecca T. Mercuri and L. Jean Camp. The Code of Elections. *Commun. ACM*, 47(10):52–57, 2004. ISSN 0001-0782. 2

Lilian Mitrou, Dimitris Gritzalis, Sokratis Katsikas, and Geradl Quirchmayr. *e-voting: Constitutional and legal requirements and their technical reflection*, chapter 4. Kluwer Academic Publishers, 2003. 29, 31

Shoichi Morimoto, Shinjiro Shigematsu, Yuichi Goto, and Jingde Cheng. A security specification verification technique based on the international standard ISO/IEC 15408. In *SAC*, pages 1802–1803. ACM, 2006. 25

Shoichi Morimoto, Shinjiro Shigematsu, Yuichi Goto, and Jingde Cheng. Formal Verification of Security Specifications with Common Criteria. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1506–1512, New York, NY, USA, 2007. ACM Press. 25

S. Myagmar, A. Lee, and W. Yurcik. Threat Modeling as a Basis for Security Requirements. In *StorageSS '05: Proceedings of the 2005 ACM workshop on Storage security and survivability*, pages 94–102, New York, NY, USA, 2005. ACM Press. 1, 4

Sebastian Olbrich and Carlo Simon. Process Modelling towards e-Government — Visualization and Semantic Modelling of Legal Regu-

lations as Executable Process Sets. *Electronic Journal of e-Government*, 6, 2008. 19

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. 3

Sylvia Osborn. Mandatory Access Control and Role-Based Access Control Revisited. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 31–40, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-985-8. 28

S. Owre, N. Shankar, and J. M. Rushby. *The PVS Specification Language*. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993. 7, 160, 190

Nathanael Paul and Andrew S. Tanenbaum. The Design of a Trustworthy Voting System. In *ACSAC*, pages 507–517. IEEE Computer Society, 2009a. 38

Nathanael Paul and Andrew S. Tanenbaum. Trustworthy Voting: From Machine to System. *IEEE Computer*, 42(5):23–29, 2009b. 38

James L. Peterson. Petri Nets. *ACM Comput. Surv.*, 9(3), 1977. 16

Alexander Prosser, Robert Kofler, Robert Krimmer, and Martin Karl Unger. Security Assets in E-Voting. In Alexander Prosser and Robert Krimmer, editors, *lectronic Voting in Europe*, volume 47 of *LNI*, pages 171–180. GI, 2004. ISBN 3-88579-376-8. 3

Frank Puhlmann and Mathias Weske. Using the i-Calculus for Formalizing Workflow Patterns. In *Business Process Management*, pages 153–168, 2005. 16

Indrajit Ray, Indrakshi Ray, and Natarajan Narasimhamurthi. An Anonymous Electronic Voting Protocol for Voting Over The Internet. In *WECWIS '01: Proceedings of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS '01)*, page 188, Washington, DC, USA, 2001. IEEE Computer Society. 37

Indrakshi Ray, Robert France Na Li, and Dae-Kyoo Kim. Using UML to Visualize Role-Based Access Control Constraints. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 115–124, New York, NY, USA, 2004. ACM. 29

U.S.A. National Performance Review. Executive summary — creating a government that works better and costs less. http://govinfo.library.unt.edu/npr/library/nprrpt/annrpt/redtpe93, 1993. 19

Ronald L. Rivest and John P. Wack. On the notion of "Software Independence" in Voting Systems, 2006. URL `http://vote.nist.gov/SI-in-voting.pdf`. 3

Russ Rogers, Greg Miles, Ed Fuller, and Ted Dykstra. *Security Assessment: Case Studies for Implementing the NSA IAM.* Syngress Publishing, 2004. 1, 27

Winston W. Royce. Managing the development of large software systems: concepts and techniques. In *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, pages 328–338. IEEE Computer Society Press, 1987. 43

Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Petia Wohed. On the suitability of UML 2.0 activity diagrams for business pro-

cess modelling. In *APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, pages 95–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920-68235-X. 15

P.Y.A. Ryan, D. Bismark, J. Heather, S. Schneider, and Zhe Xia. Prêt á voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4), 2009. 36

Gwen Salaun, Lucas Bordeaux, and Marco Schaerf. Describing and Reasoning on Web Services using Process Algebra. In 43, editor, *Web Services, IEEE International Conference on*, Los Alamitos, CA, USA, 2004. IEEE Computer Society,. 16

Ioan Salomie, Tudor Cioara, Ionut Anghel, Mihaela Dinsoreanu, and Tudor Ioan Salomie. Workflow Models Enhanced with Process Algebra Verification for Industrial Business Processes. In *ICCOMP'07: Proceedings of the 11th WSEAS International Conference on Computers*, pages 502–507, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS). 16

Altair O. Santin, Regivaldo G. Costa, and Carlos A. Maziero. A Three-Ballot-Based Secure Electronic Voting System. *IEEE Security and Privacy*, 6(3):14–21, 2008. 8, 37

Naveen Sastry, Tadayoshi Kohno, and David Wagner. Designing Voting Machines for Verification. In *Proceedings of the 15th conference on USENIX Security Symposium*, volume Volume 15, Berkeley, CA, USA, 2006. USENIX Association. 2, 8, 38

Naveen K. Sastry. *Verifying Security Properties in Electronic Voting Machines*. PhD thesis, EECS Department, University of California, Berke-

ley, May 2007. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-61.html. 10, 29, 38, 151, 160

Tim Schattkowsky and Alexander Forster. On the pitfalls of uml 2 activity modeling. In *MISE '07: Proceedings of the International Workshop on Modeling in Software Engineering*, page 8, Washington, DC, USA, 2007. IEEE Computer Society. 15

B. Schneier. Attack trees. *Dr. Dobb's Journal*, 1999. URL http://www.schneier.com/paper-attacktrees-ddj-ft.html. 21

Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2004. 21

Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated Generation and Analysis of Attack Graphs. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 273, Washington, DC, USA, 2002. IEEE Computer Society. 21

Oleg Mikhail Sheyner. *Scenario Graphs and Attack Graphs*. PhD thesis, Computer Science Department, Carnegie Mellon University, School of Computer Science Computer Science Department Carnegie Mellon University Pittsburgh, PA, 2004. 21

Lan Sommerville. *Software engineering (5th ed.)*. Addison Wesley Longman Publishing Co., Inc., 1995. 48

SRI. PVS Specification and Verification System. URL http://pvs.csl.sri.com/. 7, 160

Jan Steffan and Markus Schumacher. Collaborative Attack Modeling. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 253–259, New York, NY, USA, 2002. ACM Press. 21

Cynthia Sturton, Susmit Jha, Sanjit A. Seshia, and David Wagner. On Voting Machine Design for Verification and Testability. In Ehab Al-Shaer and Somesh Jha and Angelos D. Keromytis, editor, *ACM Conference on Computer and Communications Security*, pages 463–476, 2009. 4, 40, 41

Marcel Thaens, Victor Bekkers, and Hein van Duivenboden. Business Process Redesign and Public Administration: a Perfect Match? In *Taylor, J.A., Snellen, I.Th.M. and Zuurmond, A. (Eds.): Beyond BPR in Public Administration: An Institutional Transformation in an Information Age*, pages 15–36. IOS Press, Amsterdam, 1997. 19

Donald E. Thomas and Philip R. Moorby. *The VERILOG Hardware Description Language.* Kluwer Academic Publishers, Norwell, MA, USA, 1991. 41

Roberto Tiella, Adolfo Villafiorita, and Silvia Tomasi. Specification of the Control Logic of an eVoting System in UML: the ProVotE experience. *Proceedings of the 5th International Workshop on Critical Systems Development Using Modeling Languages (CSDUML 2006)*, pages 84–94, October 1 2006. ISSN 0809-1021. 40, 41, 45

George Valiris and Michalis Glykas. Critical review of existing BPR methodologies: The need for a holistic approach. In 5, editor, *Business Process Management Journal*, pages 65 – 86. MCB UP Ltd, 1999. 14

Axel van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *5th IEEE International Symposium on Requirements Engineering (RE 2001)*, page 249, Washington, DC, USA, 2001. IEEE Computer Society. 22

Axel van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. In *ICSE '04: Proceedings of the 26th Interna-*

tional Conference on Software Engineering, pages 148–157, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0. 22

Monika Vetterling, Guido Wimmel, and Alexander Wisspeintner. Secure Systems Development Based on the Common Criteria: the PalME Project. *SIGSOFT Softw. Eng. Notes*, 27(6):129–138, 2002. ISSN 0163-5948. 24

Monika Vetterling, Guido Wimmel, and Alexander Wisspeintner. A Graphical Approach to Risk Identification, Motivated by Empirical Investigations. *Lecture Notes in Computer Science*, pages 574–588, Thursday, November 23 2006. ISSN 0302-9743. doi: 10.1007/11880240_40. 2, 25

Adolfo Villafiorita, Andrea Mattioli, and Komminist Weldemariam. Managing Requirements for e-voting Systems: Issues and Approaches Motivated by a Case Study. In *RE-VOTE*. IEEE Computer Society, 2009a. 11, 34, 41, 43, 45, 70, 151

Adolfo Villafiorita, Komminist Weldemariam, and Roberto Tiella. Development, Formal Verification, and Evaluation of an E-Voting System With VVPAT. *IEEE Transactions on Information Forensics and Security*, 4 (4), 2009b. 11, 40, 41, 82, 154

Adolfo Villafiorita, Komminist Weldemariam, Angelo Susi, and Alberto Siena. Modeling and Analysis of Laws using BPR and Goal-oriented framework. In *To Appear In Proceedings of International Conference on Technical and Legal Aspects of the e-Society (CYBERLAWS 2010)*. IEEE Computer Society, 2010. 81, 208

Melanie Volkamer. *Evaluation of Electronic Voting: Requirements and Evaluation Procedures to Support Responsible Election Authorities*. Springer Publishing Company, Incorporated, 2009. 9, 29, 34, 35, 36, 208

Melanie Volkamer and Margaret McGaley. Requirements and Evaluation Procedures for eVoting. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 895–902, Washington, DC, USA, 2007. IEEE Computer Society. 9, 160, 208

Fredrik Vraalsen, Mass Soldal Lund, Tobias Mahler, Xavier Parent, and Ketil Stølen. Specifying Legal Risk Scenarios Using the CORAS Threat Modelling Language. In *iTrust*, pages 45–60, 2005. 2, 25

David Wastell, P. White, and P. Kawalek. A Methodology for Business Process Redesign: Experiences and Issues. *Journal of Strategic Information Systems*, 3:23–40, 1994. 14, 19

Komminist Weldemariam, Richard A. Kemmerer, and Adolfo Villafiorita. Formal Analysis of Attacks for E-voting System. In *CRiSIS '09: Forth International Conference on Risks and Security of Internet and Systems*. IEEE, 2009. 40

Komminist Weldemariam, Richard A. Kemmerer, and Adolfo Villafiorita. Formal Specification and Analysis of an e-Voting System. In *The 5th International Conference on Availability, Reliability and Security (ARES 2010)*. IEEE Computer Society, 2010. 40

Leslie Willcocks, W. Currie, and S. Jackson. Radical Re-Engineering and Information Systems: Evidence from UK Public Services. In *Proceedings of The Fifth European Conference In Information Systems*, 1997. 19

Subashish Guha William J. Kettinger, James T. C. Teng. Business Process Change: A Study of Methodologies, Techniques, and Tools. *MIS Quarterly*, 21, 1997. URL `http://www.misq.org/archivist/vol/no21/issue1/vol21n1art3.html`. 14

Guido Oliver Wimmel. *Model-Based Development of Security-Critical Systems*. PhD thesis, Institut fü r Informatik der Technischen Universitä München, February 2005. 4, 28

Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Analysis of web services composition languages: The case of bpel4ws. In *Conceptual Modeling (ER 2003)*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003. 16

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, 2004a. ACM Press. 9, 33

Alexandros Xenakis and Ann Macintosh. Levels of Difficulty in Introducing e-Voting. In *EGOV*, pages 116–121, 2004b. 33

Alexandros Xenakis and Ann Macintosh. G2G Collaboration to Support the Deployment of e-Voting in the UK: A Discussion Paper. In *EGOV*, Lecture Notes in Computer Science, pages 240–245. Springer, 2004c. 33

Alexandros Xenakis and Ann Macintosh. Procedural Security and Social Acceptance in E-Voting. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 5*, page 118.1, Washington, DC, USA, 2005a. IEEE Computer Society. 9, 33

Alexandros Xenakis and Ann Macintosh. Using Business Process Reengineering (BPR) for the Effective Administration of Electronic Voting. *The Electronic Journal of e-Government*, 3(2), 2005b. 32

Alexandros Xenakis and Ann Macintosh. A Methodology for the Redesign

of the Electoral Process to an e-electoral process. In *Int. J. Electronic Governance*, volume 1, pages 4 –16, 2007. 32

Zhe Xia, Steve A. Schneider, James Heather, and Jacques Traoré. Analysis, Improvement and Simplification of Prêt à voter with Paillier Encryption. In *EVT'08: Proceedings of the conference on Electronic voting technology*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association. 36

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. ISBN 1-59593-993-4. 4, 100

Ka-Ping Yee. Extending Prerendered-Interface Voting Software to Support Accessibility and other ballot features. In *EVT'07: Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, pages 5–5, Berkeley, CA, USA, 2007. USENIX Association. 8, 38

Yijun Yu, Haruhiko Kaiya, Hironori Washizaki, Yingfei Xiong, Zhenjiang Hu, and Nobukazu Yoshioka. Enforcing a security pattern in stakeholder goal models. In *QoP '08: Proceedings of the 4th ACM workshop on Quality of protection*, New York, NY, USA, 2008. ACM. 22

Zhan-Haomin, Yin-Guisheng, and Sun-Changsong. Process Algebra Based for Requirement Process Reorganization. In *Computer Science and Software Engineering, International Conference on*, pages 601–604, Los Alamitos, CA, USA, 2008. IEEE Computer Society. 16

Appendix A

Sample Attack Specifications

The complete original specification of ES&S e-voting system can be downloaded from the `http://ict4g.fbk.eu/people/sisai/specs/astral-specs/ESandS_Original.spec`. And, the extended version of the specification can be downloaded from `http://ict4g.fbk.eu/people/sisai/specs/astral-specs/ESandS_Extended.spec`

We only show few of the transitions from the extended specification for the ES&S e-voting system. The below transitions specify the attack scenarios specified in the DRE process, as discussed in Chapter 5.

```
TRANSITION Attack_Change_Vote ( vc: Candidate, ac: Candidate, vType: VoterType )
       ENTRY           [ TIME : ACV_Dur ]

         /*Change Vote Attack!!*/

         /*This attack asssumes unattentive voter. This results in change of vote.*/
                 Which_Phase = During_Voting
               & Terminal_mode = voter_mode
               & vType = Unattentive
               & EXISTS R: Race
                           ( vc ISIN Displayed_Candidates ( R )
                           & ac ISIN Displayed_Candidates ( R )
                           & vc ISIN tempVoteRecord ( R )
                           & Picked ( Candidate_Name ( vc ) , Race_Title ( R )  )
                           & ~Picked ( Candidate_Name ( ac ) , Race_Title ( R )  )  )
               &
         /*voter's candidate is different from attacker's candidate.*/
                 vc ~= ac
               & EXISTS b: Button
```

```
                ( b = REVIEW
                & Button_Pushed ( b )  )
        & scrName = REVIEW_SCREEN
        & ~Review_Displayed
        & ~Vote_Changed
EXIT
          EXISTS R: Race
                ( vc ISIN Displayed_Candidates' ( R )
                & ac ISIN Displayed_Candidates' ( R )
                & vc ISIN tempVoteRecord' ( R )
                & tempVoteRecord ( R )  BECOMES  ( tempVoteRecord' ( R )  SET_DIFF   vc   )  UNION
                      ac
                & Picked ( Candidate_Name ( vc ) , Race_Title ( R )  )  BECOMES FALSE
                & Picked ( Candidate_Name ( ac ) , Race_Title ( R )  )  BECOMES TRUE
                & currentRace = R )
        &
  /*enabling RTAL to print the attacker's intension.*/
          Signal_Enabled
        & Which_Signal = Vote_Signal
        &
  /*for this candidate, print Cancelled on the RTAL tape.*/
          pickedName = Candidate_Name ( vc )
        &
  /*for this candidate, print Selected on the RTAL tape.*/
          attPickedName = Candidate_Name ( ac )
        & Vote_Changed
EXCEPT          [ TIME : ACV_Dur ]

  /*Change Vote Attack.*/

  /*This attack asssumes a Careful voter. The voter has confirmed and the Thank_You message has been
     displayed.*/
          Which_Phase = During_Voting
       & vType = Careful
       & EXISTS b: Button
                ( b = CONFIRM
                & Button_Pushed ( b )  )
       & scrName = THANKYOU_SCREEN
       & EXISTS R: Race
                ( vc ISIN Displayed_Candidates ( R )
                & ac ISIN Displayed_Candidates ( R )
                & Picked ( Candidate_Name ( vc ) , Race_Title ( R )  )  )
       &
  /*voter candidate is different from attacker candidate */
          vc ~= ac
       & Min_Display ( scrNumber )  = Display_Info ( Thank_You, NoButton )
       & ~Summary_Sent2RTAL
       & ~Vote_Changed
       & NormalVotingProcess
EXIT
          EXISTS R: Race
                ( vc ISIN Displayed_Candidates' ( R )
                & ac ISIN Displayed_Candidates' ( R )
```

```
                  & vc ISIN tempVoteRecord' ( R )
               ->    ( Picked ( Candidate_Name ( vc ) , Race_Title ( R ) )  BECOMES FALSE
                     & Picked ( Candidate_Name ( ac ) , Race_Title ( R ) )  BECOMES TRUE
                     & currentRace = R )  )
       &
  /*enabling RTAL to print the attacker's intension.*/
        Signal_Enabled
       & Which_Signal = Vote_Signal
       &
  /*for this candidate, print Cancelled on the RTAL tape.*/
        pickedName = Candidate_Name ( vc )
       &
  /*for this candidate, print Selected on the RTAL tape.*/
        attPickedName = Candidate_Name ( ac )
       & Vote_Changed
EXCEPT          [ TIME : ACV_Dur ]

  /* Complete voting process attack.*/

  /*This attack asssumes a fleeing voter. In this scenario the fleeing voter voted for the attacke's
     candidate*/
         Which_Phase = During_Voting
       & Terminal_Mode = voter_mode
       & vType = Fleeing
       & scrName = REVIEW_SCREEN
       & scrNumber = Number_Of_Race + 1
       & Now - Change ( scrNumber )  >= 10
       & EXISTS R: Race
                  ( vc ISIN Displayed_Candidates ( R )
                  & ac ISIN Displayed_Candidates ( R )
                  & Picked ( Candidate_Name ( vc ) , Race_Title ( R ) )  )
       & vc = ac
       & Review_Displayed
       & NormalVotingProcess
EXIT

  /*the attacker calls the confirmation function and complete the process.*/
        scrNumber = scrNumber' + 2
       & scrName = THANKYOU_SCREEN
       & EXISTS b: Button
                  ( b = CONFIRM
               ->   Button_Pushed ( b )  BECOMES TRUE )
       & Min_Display ( scrNumber )  BECOMES Display_Info ( Thank_You, NoButton )
       &
  /*The normal voting process is intrupted by the attacker and the DRE is not chirping for this voter.*/
        ~NormalVotingProcess
EXCEPT          [ TIME : ACV_Dur ]

  /*Fake Voter Attack.*/

  /*This attack asssumes a fleeing voter. This results in faking the fleeing voter.*/
        Which_Phase = During_Voting
       & Terminal_Mode = voter_mode
```

```
          & vType = Fleeing
          & EXISTS b: Button
                    ( b = CONFIRM
                    & Button_Pushed ( b )  )
          & scrName = THANKYOU_SCREEN
          & Min_Display ( scrNumber )  = Display_Info ( Thank_You, NoButton )
          & EXISTS R: Race
                    ( vc ISIN Displayed_Candidates ( R )
                    & ac ISIN Displayed_Candidates ( R )
                    & Picked ( Candidate_Name ( vc ) , Race_Title ( R )  )
                    & ~Picked ( Candidate_Name ( ac ) , Race_Title ( R )  )  )
          &
     /*the voter's candidate is different from the attacker's candidate.*/
            vc ~= ac
          & ~Summary_Sent2RTAL
          & ~Fleeing_Faked
          & Review_Displayed
  EXIT
            Fleeing_Faked

TRANSITION Display_ReivewPage
  ENTRY           [ TIME : S_R_Dur ]
            Terminal_Mode = voter_mode
          & Which_Phase = During_Voting
          & DRE_State = Opened
          & scrNumber = Number_Of_Race + 1
          & scrName = REVIEW_SCREEN
          & EXISTS b: Button
                    ( b = REVIEW
                    & Button_Pushed ( b )  )
          & ~Review_Displayed
  EXIT
            Review_Displayed
          & Display ( scrNumber' )  BECOMES Display_Review (  { SETDEF c: Race_Candidates_Pair ( EXISTS R: Race
                              ( c [ Contest ]  = R
                              & c [ Nominees ]  =         tempVoteRecord' ( R )  )  )  } , Screen_Buttons ( scrNumber' )  )

TRANSITION Update_TotalTally
  ENTRY           [ TIME : 1 ]
            Which_Phase = During_Voting
          & Min_Display ( scrNumber )  = Display_Info ( Thank_You, NoButton )
          & EXISTS b: Button
                    ( b = CONFIRM
                    & Button_Pushed ( b )  )
          & scrName = THANKYOU_SCREEN
          & ~Summary_Sent2RTAL
  EXIT
            FORALL C: Candidate, R: Race
                    ( C ISIN Displayed_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' - 1 )  = Display_Info ( Ballot_Not_Completed, Screen_Buttons' ( scrNumber' - 1 )  )
            THEN
                    NumberOfLogEntry = NumberOfLogEntry' + 1
                  & EventLog ( NumberOfLogEntry )  BECOMES underVotedRaces'
                  & underVotedRaces = 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
          &
    /*Reset the temprary vote record. */
            FORALL R: Race
                     ( tempVoteRecord ( R )  = EMPTY )
          & Summary_Sent2RTAL


TRANSITION Attack_ReDisplay ( DelayTime: Time )
  ENTRY           [ TIME : ACV_Dur ]
            Fleeing_Faked
          & Now - Change ( Fleeing_Faked )  >= DelayTime
  EXIT

    /*The attacker again display the confirmation page.*/
            scrNumber = scrNumber' - 1
          & scrName = CONFIRM_SCREEN
          & Min_Display ( scrNumber )  BECOMES Display_Info ( voterMessage',   BACK, CONFIRM   )


TRANSITION Attack_Call_ChirpingRt
  ENTRY           [ TIME : 1 ]
            Which_Phase = During_Voting
          & Terminal_Mode = voter_mode
          & Fleeing_Faked
          & scrNumber = Number_Of_Race + 2
          & Now - Change ( scrNumber )  >= 10
  EXIT
            Terminal_Mode = chirping
```

The following transition specifies how the RTAL prints the information sent by the DRE process. Its entry condition specifies that the DRE is connected to the RTAL, the DRE has

sent a signal to the RTAL for printing, the send signal should be in one of the permitted voting procedures (i.e., My DRE.Which Signal = NoSignal), and the RTAL is currently in waiting state. The exit specifies a complex conditional. It is related to the normal voting process as carried out by legitimate voters (careful or attentive voters) and the altered voting process due to the various attacks as discussed in the paper.

```
TRANSITION Print_Selection
     ENTRY          [ TIME : P_S_Dur ]
               My_DRE.Plugged_In
             & My_DRE.Signal_Enabled
             & RTAL_State = Wait
             & My_DRE.Which_Signal ~= NoSignal
     EXIT
               RTAL_State = Printed
             & IF
                       My_DRE.Which_Signal = Start_Signal
                    |  My_DRE.Which_Signal = Vote_Signal
               THEN
                       IF
                               My_DRE.Which_Signal = Start_Signal
                       THEN
                               tapePosition = tapePosition' + 1
                             & CutLengthCounter = CutLengthCounter' + 1
                             & voterNumber = voterNumber' + 1
                             & Tape ( tapePosition )  BECOMES Make_Print_Info ( My_DRE.RTALMessage )
                       ELSE
                               IF
                                       My_DRE.Vote_Changed
                               THEN

       /*Attack point.*/
                                       tapePosition = tapePosition' + 2
                                    & CutLengthCounter = CutLengthCounter' + 2
                                    &
       /*This prints the cancelation of the voter's candidate choice.*/
                                       Tape ( tapePosition - 1 )  = Make_Print_VoteEntry ( My_DRE.pickedName,
                                                                                          My_DRE.currentRace,
                                                                                          Cancelled )
                                    &
       /*This prints the selection of the attacker's candidate selection.*/
                                       Tape ( tapePosition )  = Make_Print_VoteEntry ( My_DRE.attPickedName,
                                                                                      My_DRE.currentRace,
                                                                                      Selected )
                                    & FORALL i: Tape_Number
                                                ( i ~= tapePosition
                                                & i ~= tapePosition - 1
                                              ->   NOCHANGE  ( Tape ( i )  )  )
                               ELSE

       /*Normal voter's choice print.*/
```

```
                        tapePosition = tapePosition' + 1
                      & CutLengthCounter = CutLengthCounter' + 1
                      & Tape ( tapePosition )  BECOMES Make_Print_VoteEntry ( My_DRE.pickedName,
                                                                             My_DRE.currentRace,
                                                                             My_DRE.pickedValue )
                FI
        FI
ELSE
        IF
                My_DRE.Vote_Changed
        THEN

/*if attacked before the summary printed. This is the case for careful voter.*/
                tapePosition = tapePosition' + 5
              & CutLengthCounter = CutLengthCounter' + 5
              & Tape ( tapePosition - 4 )  BECOMES Make_Print_VoteEntry ( My_DRE.pickedName,
                                                                         My_DRE.currentRace,
                                                                         Cancelled )
              &
/*This prints the selection of the attacker's candidate selection.*/
                Tape ( tapePosition - 3 )  BECOMES Make_Print_VoteEntry ( My_DRE.attPickedName,
                                                                         My_DRE.currentRace,
                                                                         Selected )
              &
/*followed by immediate print of the summary information and barcode.*/
                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
                          & i ~= tapePosition - 3
                          & i ~= tapePosition - 4
                        ->   NOCHANGE  ( Tape ( i )  )  )
              & VoteStartPosition ( voterNumber )  BECOMES tapePosition - CutLengthCounter +
                  1
              & VoteEndPosition ( voterNumber )  BECOMES tapePosition
              & summaryPrinted = TRUE
        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 = TRUE
        FI
    FI
```

Appendix B

Publications

Journal

[1] Adolfo Villafiorita, Komminist Weldemariam, Roberto Tiella. **Development, Formal Verification and Evaluation of an e-voting System with VVPAT**. *IEEE Transaction on Information Forensics and Security.*

Book Chapter

[2] Luca Cernuzzi, Magalí Gonzáez, Marco Ronchetti, Adolfo Villafiorita, Komminist Weldemariam. **Multi-cultural experiences in eGovernance: Case Studies and a Roadmap**. *Global Strategy and Practice of e-Governance: Examples from Around the World (Accepted for publication).*

Archival Proceedings

2010

[3] Komminist Weldemariam, Richard Kemmerer, Adolfo Villafiorita. **Formal Specification and Analysis of an e-Voting System.** *To Appear In Proceedings of The 5th International Conference on Availability, Reliability and Security (ARES 2010).* **Publisher IEEE.**

[4] Komminist Weldemariam, Adolfo Villafiorita, Angelo Susi, and Alberto Siena. **Modeling and Analysis of Laws using BPR and Goal-oriented framework.** *To Appear In Proceedings of International Conference on Technical and Legal Aspects of the e-Society (CYBERLAWS 2010).* **Publisher IEEE.**

[5] Birhanu Eshete, Andrea Mattiolli, Adolfo Villafiorita, Komminist Weldemariam. **ICT for Good: Opportunities, Challenges and the Way Forward.** *In Proceedings of The International Conference on Digital Society (ICDS 2010)* **Publisher IEEE.**

2009

[6] Aaron Ciaghi, Komminist Weldemariam, Adolfo Villafiorita, Andrea Mattioli, Quoc-Sang Phan. **Supporting Public Administration with an Integrated BPR Environment.** *To Appear In Proceedings of The 1st International ICST Conference on e-Infrastructure and e-Services for Developing Countries (AFRICOMM09).* **Publisher Springer.**

[7] Komminist Weldemariam, Richard Kemmerer, Adolfo Villafiorita. **Formal Analysis of Attacks for e-voting System**. *In Proceedings of the 4th International Conference on Risks and Security of Internet and Systems (CRiSIS 2009).* **Publisher IEEE.**

[8] Komminist Weldemariam, Adolfo Villariorita, Andrea Mattioli. **Experiments and Data Analysis of Electronic Voting System.** *In Proceedings of the 4th International Conference on Risks and Security of Internet and Systems (CRiSIS 2009).* **Publisher IEEE.**

[9] Komminist Weldemariam, Andrea Mattioli, Adolfo Villafiorita. **Managing Requirements for e-voting Systems: Issues and Approaches Motivated by a**

Case Study. *In Proceedings of the first International Workshop on Requirements Engineering for E-voting System, In conjunction with the 17th IEEE International Requirements Engineering Conference.* **Publisher IEEE.**

2008

[10] Komminist Weldemariam, Adolfo Villariorita. **Formal Procedural Security Modeling and Analysis.** *In Proceedings of the 3rd International Conference on Risks and Security of Internet and Systems (CRiSIS 2008).* **Publisher IEEE.**

[11] Komminist Weldemariam, Adolfo Villafiorita. **A Methodology for Assessing Procedural Security: A Case Study in E-Voting.** *In Proceedings of the 3rd International Conference on Electronic Voting (EVOTE 2008).* **Publisher Springer.**

[12] Komminist Weldemariam, Adolfo Villariorita. **Modeling and Analysis of Procedural Security in (e)Voting: the Trentino's Approach and Experiences.** *In Proceedings of the 3rd USENIX Electronic Voting Technology (EVT 2008).* **Publisher USENIX/ACM.**

2007

[13] Komminist Weldemariam, Adolfo Villariorita. **Assessing Procedural Risks and Threats in e-Voting: Challenges and an Approach.** *In Proceedings of the First Conference on E-Voting and Identity (VOTE-ID).* **Publisher Springer.**