A Conceptual Approach to Secure Elections Based on Patterns

Eduardo B. Fernandez¹, David L. La Red² and José I. Peláez³,

¹ Dept. of Computer Science and Eng., Florida Atlantic University, Boca Raton, FL, USA, ed@cse.fau.edu
² Computer Science Department, Northeast National University, Argentina, la-mdavid@cca.unne.edu.ar
³ Department of Languages and Computer Sciences, University of Malaga, Spain, ipelaez@uma.es

Abstract. We present a unified approach to elections based on a conceptual model built using software patterns. A set of analysis patterns describe the structure and management of an election. The patterns describe voter registration, ballot structure, election day management, voting process, and the complete election. A conceptual model is built from the analysis patterns and defines precisely the functional properties of such a system together with its security requirements. The conceptual model is defined using UML diagrams that can be extended or complemented with formal methods, e.g. OCL, to make the system verifiable. We show how to continue to the design phase for this system.

Keywords: democracy, e-government, patterns, secure elections.

1 Introduction

Elections and voting are fundamental aspects of any democracy. They must be performed in a secure way so that intruders cannot change the results of an election or violate the privacy of the voters. Almost all existing studies of voting systems mix together functional and implementation aspects or concentrate on specific parts of the voting process. We believe that when designing complex systems such as this one we need to present first a conceptual view of the whole system and then worry about implementation aspects. Without a conceptual semantic structure it is very difficult to convince ourselves that all requirements have been fulfilled. We need a clear and systematic structure, where we can apply security restrictions at each functional unit and activity. This approach must cover all aspects of an election in a unified way, going from voter registration to election results and from conceptual information models to implementation details. Object-oriented design and patterns can provide such a unified way. Patterns are encapsulated solutions to recurrent problems and have proven useful in handling complex systems with strict nonfunctional requirements.
We present here a pattern-based methodology to define a generic structure for elections. This structure is made up of individual patterns and itself can be considered a composite pattern or a reference architecture. Our approach here is an extension or particularization of a general methodology, still under development, which has proven useful for the early parts of a system lifecycle [1]. While many approaches and solutions have been proposed for voting, we believe this is the first attempt to define a systematic structure for the complete electoral process. We had defined the general requirements of a global approach to voting software in a previous paper [2], this paper is a concrete realization of some of those ideas within a specific development approach.

Our approach considers each lifecycle stage and applies appropriate analysis and security patterns to them; the analysis patterns define the functional aspects, while the security patterns define the required security mechanisms. The security mechanisms are deduced from an enumeration of threats [3]. This method results in a conceptual model built applying instances of analysis patterns with superimposed instances of security patterns. We use a specific type of analysis patterns, which we call Semantic Analysis Patterns (SAPs), where each pattern corresponds to a few important use cases [4].

To factor out variations in specific elections we build first an election domain model where security patterns realize the effect of regulations and defend against generic threats. From the secure domain model we generate applications, tailored to specific electoral processes, e.g. the U.S. Presidential Election, which must incorporate the domain model restrictions but can also include additional restrictions. In the design stage of each application platform, restrictions are applied, e.g. use of cloud computing. In this stage the secure conceptual model is converted to a software model where interfaces, distribution strategies, and persistence are considered. This language-independent model can be converted then into a language-specific model, e.g. .Net oriented. Finally the models are converted into code or are used to select COTS components. In this paper we consider only the requirements and analysis stages where we end with a secure conceptual model.

Section 2 provides requirements for elections, while section 3 shows the intents of all the patterns we have developed, including details of one of them. Section 4 considers threats to the election process and defines countermeasures. Section 5 shows discussion and related work. We end with some conclusions in Section 6.

2 Elections

Many human activities require some form of voting in order to elect one or more persons for specific positions in an institution or political unit, or to decide on specific issues. Elections must satisfy the following requirements, some of which come from [5]:

Privacy. There should be no way to link voters to their votes.

Eligibility. Only eligible voters can vote and they must be registered before the election day.

Uniqueness. Only one vote per voter should be counted.
Eduardo B. Fernandez, David L. La Red and José I. Peláez

No receipts. Voters cannot prove to a third party how they voted. This prevents vote selling or voter coercion.

Accuracy. All cast votes should be counted.

Integrity. Votes should not be modified, deleted, or forged.

Verifiability and auditability. It should be possible to verify that all votes have been counted and there should be a way to audit the election in case of complaints.

Secrecy. The issued vote is known only to the voter.

Availability. Users should have convenient access to the voting process and should be able to vote in a reasonable time.

Certifiability. The whole election process, including the hardware and software used should be certifiable according to some predefined criteria.

Cost-effectiveness. The election structure should have a reasonable cost on equipment and access to allow everybody to participate.

We can think of these requirements as policies for system design, that must be realized by the complete system. Additionally, specific elections may include more policies but the set above should be present in every election.

More specifically, each election defines some conditions for the potential voters to be eligible to vote. Before the election is held the potential voters must register. Also, before the election the format of the ballot describing the candidates and issues must be defined and published. Managing the election day requires having the proper workers and the appropriate number of voting machines or devices; that is, a physical infrastructure to allow the voters to properly cast their votes. A voting process lets the users vote, collects their votes, and tallies the results of the election. We have produced patterns for all those aspects and we collected them into an election pattern [6]. This latter is an example of a Semantic Analysis Pattern (SAP) [4] that corresponds to a larger reusable semantic unit. Fig. 1 shows a pattern diagram for these patterns. This diagram shows the election process pattern as using four patterns. We show parts of one of those patterns in the following sections and we end with the election pattern itself. The terms used for elections vary according to country, institution, and type of election; whenever we could not find a generic term we used a term based on our experience with elections in the USA.

3 Patterns

3.1 Voter Registration

This pattern describes the process of keeping track of who is eligible to vote. Its main objective is to allow voters to vote exactly once in each election.

Context

A political or institutional system, where we need to elect some candidates and/or decide on some issues. The system may be large enough to require division into several voting units.
Problem
In an election, every voter must be allowed to vote exactly once. Voters may try to take advantage of multiple residences or multiple identifications to vote more than once.

The solution to this problem is affected by the following forces:

- The registration process should be simple; otherwise many potential voters would not register.
- We need a way to keep track of all voters; otherwise, voters may not be able to vote.
- There may be many voters and the registration process should be scalable.
- We need to prevent that voters register more than once; otherwise they could vote more than once.
- Voters should be assigned to the voting units more convenient for them to vote, e.g., those closer to their residences.
- When voters change their addresses they may need to be assigned to a new voting unit.
- Each voting unit should know who is allowed to vote in it.
- There is a period before an election when registration is not possible.

Solution
Keep local voter lists at each voting unit and collect them into a global list. Every time a new voter tries to register check if she is already registered and if not, assign her to a voting unit.

Structure
See Fig. 2. A GlobalVoterList is created that collects all newly registered voters. A VoterList is created each time a new VotingUnit is created. The VoterList is a collection of Voters. A Voter is created when a person is eligible to vote and has not registered before.
The pattern template includes also dynamics, known uses, consequences, and related patterns. For the other patterns we show only their intent.

Fig. 2. Class diagram for Voter Registration.

3.2 Election Ballot Definition

This pattern describes how to define the structure of a ballot to be used in an election. The ballot may consist of selections of more than one position and/or issue. There can be several subsets of the general ballot depending on the location.

3.3 Election Day Management

This pattern describes the management of the infrastructure required to provide the voters an appropriate place of voting during the election period. Typically, this includes the handling of voting machines (devices) distributed at different locations. It also includes maintaining the voter list in each voting unit. Finally, it describes the assignment of employees to voting units.

3.4 Voting Process

This pattern describes the voting process in which voters cast their votes in a secret manner and only once. The votes are recorded in such a way that the results of an election can be obtained using different viewpoints, e.g. the overall result or the result for each voting at each location.
3.5 Election Process

This pattern combines the previous patterns, including voter registration, ballot structure definition, election day management, and voting process. Fig. 3 shows its class structure. Fig. 4 shows the activity diagram of the overall process of an election management. Assume that we already have a list of locations and lists of voters (a global voter list and local voter lists for each location). Thus, we first decide which positions and/or issues are to be voted on and who the candidates are. Then, we should be able to define a general ballot which includes the positions and candidates that will be voted at all locations, e.g. a position for the President. Meanwhile, we define which locations should participate in the election, which will then be activated accordingly with the voter lists for the selected locations.

Fig. 3. Class diagram for Election Process.
Fig. 4. Activity diagram for an election.
Once a general ballot is defined, we can define local ballots as well as assign personnel and voting machines to each location. Existing employees are activated and notified while new employees are assigned to locations and notified. When the election period starts, voters cast their votes which are kept at their location. After the election period ends, each location counts their votes and produces their local results. These results will be aggregated to create a global result, which then will be published.

There are several activities that can be added such as:

- We can add/remove locations to/from the election, accompanied with voter lists for those locations.
- More people may register to vote, and then added to the voter list of the location at which they register.
- We can add/remove candidates or issues from the ballots.
- We can tailor ballots for local elections which may have other issues or additional candidates.
- In case that there is any protest of the result for a location, votes in that location must be recounted, normally by the officers from election management.

The object-oriented approach allows all these changes to be easily accommodated by adding, removing, or modifying classes.

4 Enumerating Threats and Defenses

A clear advantage of this conceptual approach is that we can systematically and rather completely enumerate threats [3]. We can analyze each activity in the activity diagram that describes the business flow of the application and see which attacks are possible according to the goals of the attacker. This is much simpler and clearer than try to find out low-level attacks such as buffer overflows or other implementation-related attacks. Of course, the code of the final system must still be analyzed for those attacks but a good early design eliminates ranges of attacks and if they happen it will mitigate their effects. For example, in Fig. 4, the activity Vote can be attacked in the following ways (threats Tvi):

- Tv1: Voter is an impostor.
- Tv2: Vote is modified or deleted after being cast.
- Tv3: Vote is associated to a specific voter.
- Tv4: Voter is coerced.
- Tv5: Voter obtains copy of her vote.
- Tv6: Voter cannot vote.

These threats were obtained using as a reference the requirements we described earlier. Their enumeration requires knowledge of the voting process, not of security. We can do the same for each activity and we can obtain in this way almost all of the threats, depending on the experience and knowledge of the enumerator. Once we have enumerated all the threats we can determine security policies to stop or mitigate these attacks. For example, if the voter could be an impostor, we can require the voter to be
authenticated. These policies lead to security patterns [1]. The conceptual stage (domain model or analysis model) ends with a conceptual model where the appropriate patterns are added to the conceptual model.

We can go to the design stage by converting the conceptual classes of the analysis model into design classes that correspond to software artifacts. We can refine, for example, the Voting Machine as shown in Fig. 5. Every refinement should help comply with the policies defined earlier; for example:

- **User interface.** Displays ballot of the voter and after she casts her vote, confirms the indicated selection. This corresponds to the policy of vote confirmation.

- **Vote storage.** The accumulated totals are stored to comply with the policy of count integrity.

- **Paper trail.** A paper vote is created, which is stored in the voting place and cannot be carried by the voter. This complies with the policies of audit and no-receipts.

As indicated, this level would add new threats:

1. A voting machine can be replaced by a tampered machine.
2. A voting machine may be attacked by viruses, worms, or Trojan horses.
3. A voting machine may receive an attack through its communication lines or its power lines.

Now we need additional policies to stop the new attacks:

1. Hardware authentication and physical access control to avoid machine replacement.
2. Install and keep up-to-date anti-virus, anti-spyware, and anti-rootkit software. Implement routers that filter according to the Appendix A of CA-96.21 – TCP SYN Flooding and IP Spoofing Attacks [7]. Disable unnecessary web services. Install firewalls and IDS. Add redundancy and cryptography to protect the data [5] [8].

3. Implement redundant communication lines and redundant power supplies [5].

Kohno et al. analyzed the voting machines used in the US election of 2001 [9]. They complained that the code was not well documented but did not ask for modeling and verification of the complete system. [10] discussed how to design voting machines so that they can be formally verified and systematically tested. Their approach proves correctness of functions but does not consider intentional attacks to the machine. However, their approach could be combined with ours to produce a machine that can be shown also to be secure.
5 Discussion and Related Work

We mentioned above some work on design for voting machines. While this work is certainly important, analyzing voting machines in isolation is not enough, we need a holistic solution and we list below some attempts at complete systems.

A SOA-Based e-voting system is described in [10]. However, SOA is just one possible implementation architecture; also, it does not provide a conceptual model to convince ourselves that the system does follow the required policies. The author claims it is scalable and portable; however, it is not clear if it is reusable.

[11] describes the use of ID-card-based PKI for voting from mobile devices in Estonia in recent elections and suggests the use of cell-phone voting based on similar principles. However, this is just one of several possible implementations and the approach does not provide a general and unified approach to voting.

[12] considers a voting system as made up of a set of components and proposes to do threat analysis and verification for each component. Components are units of deployment and as such are a design concept. We think security must start at an earlier stage.
[13] describes the implementation of a complete voting system. The design, however, is rather ad hoc, handling independently each issue. While interesting, the results don’t appear very generalizable.

Another complete design is Helios [14]. This system is available in the Internet and can be used for actual elections. Again, its design is based on handling independently each issue and providing a good solution for it but there is no attempt of modeling or conceptualizing the whole system.

These designs (and others we saw) emphasize the correctness and tamper-free properties of their systems but don’t provide a systematic analysis of security threats for the different uses of the system and most of them are not reusable or generalizable.

6 Conclusions

We have presented a unified approach to describing the requirements for the structure and process of elections. We have produced a conceptual model that defines precisely the functional properties of such a system. We analyzed systematically its security threats in order to define its security requirements. The resulting secure conceptual model is defined using UML diagrams that can be extended or complemented with formal methods, e.g. OCL, to make the system verifiable. These requirements and the model are scalable and reusable, a clear advantage over ad hoc designs. The models are also very adaptable, we can easily tailor them for specialized elections by removing or adding classes or patterns. We showed how to continue from the requirements to the design phase for this system. We are considering a design based on components based on some of the ideas in [12]. An important aspect we did not consider was reliability and we are studying it now.

References

2. La Red, D., Peláez, J.I.: La Ingeniería Global del Software en los Procesos Electorales. 31 JAIIO (Argentine Conference on Computer Science and Operational Research) - AST 2002 (Argentine Symposium on Computing Technology); ENARS (Encuentro de Administradores de Redes y Sistemas); Anales, págs. 28—37, Argentina (2002)
   http://drops.dagstuhl.de/opus/volltexte/2008/1300 (last retrieved Sept. 03, 2010)