Definition and Analysis of Election Processes

Mohammad S. Raunak, Bin Chen, Amr Elssamadisy,
Lori A. Clarke, Leon J. Osterweil

Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
{raunak, chenbin, samadisy, clarke, ljo}@cs.umass.edu

Abstract. This paper shows that process definition and analysis technologies can be used to reason about the vulnerability of election processes with respect to incorrect or fraudulent behaviors by election officials. The Little-JIL language is used to model example election processes, and various election worker fraudulent behaviors. The FLAVERS finite-state verification system is then used to determine whether different combinations of election worker behaviors cause the process to produce incorrect election results or whether protective actions can be used to thwart these threats.

1 Introduction

In previous work, we have demonstrated that it is possible to define complex processes with precision that is sufficient to support definitive demonstrations that the processes either do, or do not, have worrisome defects. Our preliminary work with healthcare processes [3], for example, shows that it is possible to identify potentially life-threatening defects, even in large complex medical processes. Our work with the US National Mediation Board has suggested that automating carefully defined processes that have been clearly understood by all stakeholders, can lead to increased trust and confidence in the workings of government.

This paper extends our previous process definition and analysis work to election processes. A novel aspect of this work is its approach to assessing the potential impact of fraudulent behavior. In our earlier work (e.g. with healthcare processes [3]) we assumed that participating agents (e.g. doctors and nurses) always try to perform assigned tasks correctly. We dealt with incorrect or inadequate performance through the use of pre- and post-condition checks and exception processing. But, in analysis of elections, we now attempt to deal with the consequences of the performance of tasks by agents whose actions may be intentionally incorrect or malicious. An interesting challenge of this work is how to represent such behaviors and assess how well processes defend against their negative effects. Early positive results of this work suggest the possibility of a discipline of election process engineering, in which costs and benefits of specific safeguards can be measured against specific election fraud risks.

2 Related Work

There is now considerable interest in assuring the correct performance of elections. The 2000 US Presidential election yet again demonstrated that elections may have many and varied defects and loopholes [1], [4], [6], [8]. In response there have been many efforts to improve the conduct of elections [7], [9]. Most efforts focus on using electronic devices to record and tabulate votes and emphasize the potential for such devices to commit errors or frauds. Because electronic voting machines use software for vote recording and tabulation, software analysis is used to reason about the code in such machines. Our work differs in that it seeks to discover and correct defects in the overall processes of which voting machines are only a part. Elections generally have many different steps and activities, and are performed by many different agents (e.g. the voter, precinct officials, and district voting officials) in addition to just the actual vote recording device. Thus, opportunities for frauds and errors go far beyond those that can be accomplished by the software code in a voting machine. Our work employs analysis techniques and approaches that are applicable to reasoning about software code, and applies these techniques instead to rigorous definitions of overall election processes to demonstrate the presence or absence of specified defects and the resistance (or its lack) in specific processes to specific frauds.

3 Our Approach

For this research we used our process language, Little-JIL [2], [10] to define an election process. We used the resource specification and dataflow annotation features of Little-JIL to represent artifact flow and agent binding details in the process definition. We specified election security requirements as finite state automata, and then used our FLAVERS finite-state verification system [5] to identify vulnerabilities and to prove whether a process can defend against a particular type of fraudulent behavior or threat.

3.1 The Little-JIL process language

The Little-JIL language supports defining coordination amongst human and automated agents at different abstraction levels. It supports the definition of control flow, including the handling of exceptions, and it also supports the definition of artifacts and their flows. The central construct of a Little-JIL process is a step. Steps are organized into a hierarchy, whose leaves represent the smallest specified units of work, each of which is assigned to an agent.

Figure 1 shows the graphical representation of a Little-JIL step with its different badges and possible connections to other steps. The interface badge specifies artifacts either required for, or generated by, the step's execution as well as the resources needed to support step execution. Every step has a special resource, its 'agent', which

is responsible for the step's execution. A step may also include pre- and/or post-requisite badges, representing steps that need to be executed before and/or after this step. On the left, inside the central black box of every non-leaf step, is a control flow badge that specifies the order in which the step's sub steps are to be executed. A child is connected to its parent by an edge, and artifact flows between the parent and child are indicated by annotations on this edge. On the right of the step bar is an X sign that represents the exception handling capabilities of the step. Attached to this badge by exception edges are handlers that deal with exceptions occurring in the step's descendants. Each handler is itself a step annotated to indicate the type of exception it handles. One of four exception continuation semantics define how process flow continues.

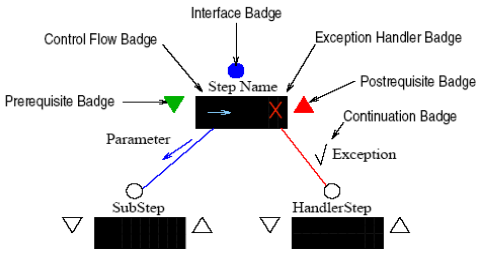

Fig. 1. A Little-JIL step construct

There are four different non-leaf *step kinds*, namely "sequential", "parallel", "try" and "choice". Children of a "sequential" step are executed one after another from left to right. Children of a "parallel" step can be executed in any order, including in parallel. A "try" step attempts to execute its children one by one from left to right. A "choice" step's agent chooses which of its children will comprise the step's execution.

A complete Little-JIL process definition also contains definitions of *artifacts* and *resources* to complement this coordination definition. *Artifacts* are entities such as data items, files, or access mechanisms that are passed between parent and child steps, much in the same way that parameters are passed in a procedure invocation in a standard programming language. Complete details about Little-JIL can be found in [10].

3.2 Process verification

We used a finite-state verifier (our FLAVERS tool) to determine whether or not election soundness policies are violated by our election process definition. Given a property that represents a policy in terms of the states of process steps and the artifacts flowing between the steps, FLAVERS determines if this property always hold on all possible process executions. When properties may not hold for all executions, FLAVERS provides a counterexample execution showing where a violation occurs, thereby providing process-improvement guidance.

4 An Election Process Example

Our election process assumes that one single DRE (direct recording electronic) voting machine is being used at a precinct, that there is only one office for which an election is being held, and that there are two candidates (A and B) running for the office.

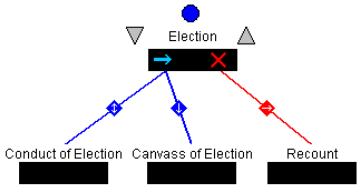

Fig. 2. Top level election process

At the top level, as shown in figure 2, the state-wide election process consists of *Conduct of Election*, followed by *Canvass of Election*, where state board officials aggregate precinct level election results, and a possible *Recount* if something major goes wrong. The child steps of the root step (*Election*) are elaborated in separate diagrams (e.g. Fig. 3 elaborates the details of *Conduct of Election*). The diagrams use yellow post-it notes to provide still more detailed elaboration.

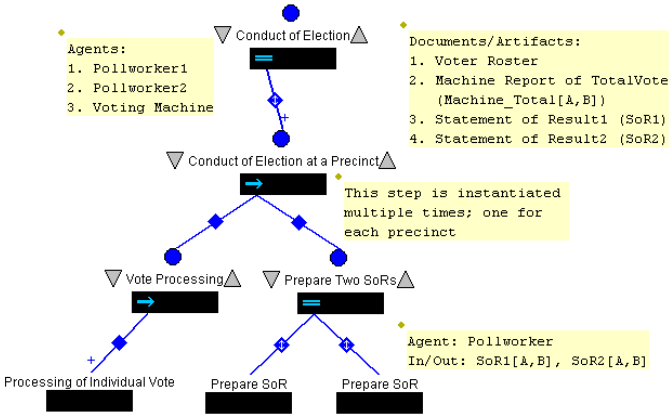

Fig. 3. Process model for *Conduct of Election*

Conduct of Election is a number of parallel activities taking place simultaneously at each precinct. *Conduct of Election at a Precinct* includes the processing of individual votes for each voter throughout the voting period and preparation of two copies of a precinct result summary called "Statement of Result" (SoR). *Processing of Individual Vote* is a reference to a step, which we have not elaborated in this paper due to space constraints. In that subprocess, an individual voter is first authenticated before being allowing to vote. A DRE is responsible for recording voter's exerted intent correctly, and the DRE keeps a running tally of the number of votes cast for each candidate.

At the end of the voting period, each of the two poll workers independently looks at the voting machine, and prepares a "Statement of Result" (SoR), consisting of the

total votes for candidate A (Machine_Total[A]) and for candidate B (Machine_Total[B]). The two SoRs are then sent to the State Election Board for statewide aggregation and certification in *Canvass of Election*. Note that *Prepare SoR* is a task assigned to a poll worker agent. The agent can carry out the task honestly or may inadvertently or maliciously modify the numbers while preparing the SoR.

The State Election Board collects the precinct level result summaries (SoRs), validates the results reported in SoRs by matching them against each other, and aggregates the precinct summaries into a statewide summary-sheet that holds the total votes for candidates A and B. The State Election Board officials make sure that the totals reported in SoR1 match the totals reported in SoR2. If there is a mismatch, the officials examine the actual DRE to determine if one of the SoRs agrees with the machine. If so, the other SoR is corrected accordingly. These actions are part of the agent behavior represented by the *Handle Validation Fail* exception handling step. If neither SoR agrees with the DRE, the precinct officials are called in for consultation and both SoRs are corrected (these details are omitted from this example because of lack of space). Upon proper aggregation of the precinct results, including handling of any potential inconsistencies in the Statement of Results, the state board certifies the result and declares it official. Figure 4 shows the model for the last part of this process. The figure also shows how the Little-JIL system's inspector tools supports looking inside a step (*Statewide Aggregation* in this case) to reveal its resource and artifact definitions.

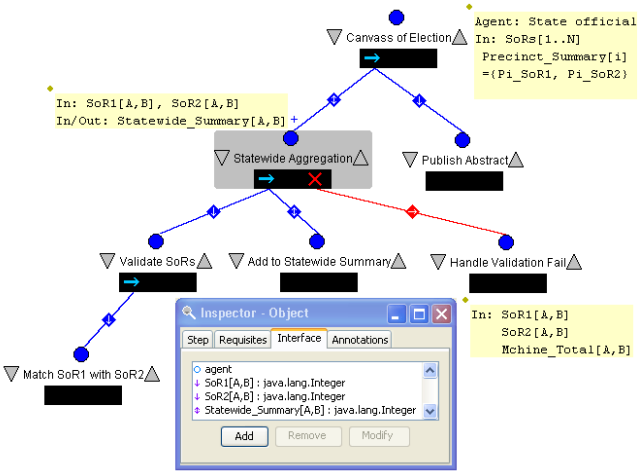

Fig. 4. Process model describing *Canvass of Election*

4.1 Analysis of Frauds

To analyze potential threats arising out of fraudulent agent behavior we use Little-JIL to model not only an election process, but also details of how an agent carries out tasks in this process. The analysis is applied to the process definition, potentially

paired with different agent behavior definitions to see which properties hold when different processes are paired with different agent behaviors.

To demonstrate this, we consider the following property for our election process: *If two SoRs mismatch, the incorrect SoR gets detected and corrected before getting added to the Statewide Summary*. For this analysis, we paired the process described in section 4.1 with specifications of the behaviors of two different poll workers, exactly one of which is hypothesized to produce fraudulent SoR results. Space limitations prevent us from showing how we used Little-JIL to model the agents' performance of the *Prepare SoR* step. But, informally, we defined the fraudulent behavior by indicating that the observed machine vote totals were redefined by the poll worker. Our model of the performance of the correct poll worker showed that the totals reported were unchanged from the original machine totals.

Our process verification framework proved that the above property holds for this process/agent pairing. It also holds when both poll workers are honest. If we were to pair this process with two dishonest agents producing identical, yet incorrect SoRs, the process verification still proves, albeit misleadingly, that the property holds. This means that this property is not sufficient for verifying this additional incorrect agent behavior. This led us to develop a stronger property: *An SoR will never get added to the Statewide Summary if it is different from the Machine_Total.* A subsequent analysis showed that this new property is sometimes violated by the process when there are two dishonest poll workers. This led us to improve the process to have safeguards against this additional incorrect agent behavior.

In this iterative process improvement procedure, we specify a property, verify it holds for a process, specify a stronger property, attempt to verify the new property to identify where it fails and improve the process with additional safeguards and prove the strength of the new process through the verification of the new property.

In what follows we describe the detail of how the property verification works in our mechanism through the example of verifying the first property mentioned above. The formalism we used for specifying this property is a finite state automaton (FSA) shown in figure 5. For each poll worker, the following property should hold.

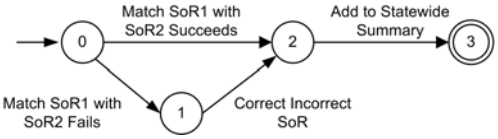

Fig. 5. A security property for the Election process

The labels associated with the transitions in this property representation automaton correspond to events in the process. The execution of a Little-JIL step is an event, but only some step execution events are germane to the property being analyzed. In the above property, the event "Match SoR1 with SoR2 succeeds" represents the execution of step *Match SoR1 with SoR2* is completed, while the event "Match SoR1 with SoR2 fails" represents the execution of step *Match SoR1 with SoR2* is terminated raising some exception. Similarly, the "Correct Incorrect SoR" transition represents execution of the step *Handle Validation Fail* is completed. Tools such as the

FLAVERS system used here trace all possible paths through a process model and move the automaton from state to state as events in the automaton alphabet are encountered along a path. If the automaton is always in an accepting state after tracing all possible paths, then the property is verified. If not, it means FLAVERS has discovered a process execution path along which the property is violated.

We now show that for one honest poll worker and one dishonest poll worker, the property specified above holds. Since only one poll worker changes the SoR, two SoRs are not identical. Therefore when the execution goes to *Match SoR1 with SoR2*, exception "ValidationFail" will be thrown. The event "Match SoR1 with SoR2 fails" occurs and the property automaton goes to state 1. The exception is then handled by the exception handler *Handle Validation Fail*. When *Handle Validation Fail* step completes, the event "Correct Incorrect SoR" occurs and leads the property to state 2. Since the exception handler is a continue handler, the step *Add to Statewide Summary* will be executed. Completion of this step triggers the event "Add to Statewide Summary", which drives the property to state 3. The property remains in the accepting state 3 until the whole process completes. Thus the property holds.

The property may also hold if both poll workers are dishonest. When both poll workers change their SoRs, two SoRs could either be identical or different. If they are identical, when execution goes to step *Match SoR1 with SoR2*, no exception will be thrown. Thus event "Match SoR1 with SoR2 Succeeds" occurs and the property goes to state 2. Then *Add to Statewide Summary* step is executed and event "Add to Statewide Summary" occurs. The property goes to the accepting state 3 and remains in this state until execution ends. On the other hand, if two SoRs are different, the "ValidationFail" exception will be thrown by the step *Match SoR1 with SoR2*. The rest of the execution is the same as the one for only one dishonest poll worker, as shown above. In this case, the property is also in the accepting state when the execution ends. Thus the property may hold if both poll workers are dishonest.

The original property only detects a fraud if the poll workers produce different SoRs. Therefore the current process and analysis are inadequate to detect a fraud where the poll workers change both SoRs in the same way. We have to modify the process and/or include a more complex property to check for this kind of fraud. While it may be possible to verify these properties by careful inspection, we argue that such manual inspection quickly becomes infeasible as the process and agent behaviors grow larger and more complex.

5 Conclusion and Future Work

This paper describes how rigorous process definition and analysis can identify vulnerabilities introduced by agent behaviors. We verified an important property for a particular election process and a specific combination of agent behaviors, and indicated how to iteratively improve the process to make it robust against more complicated fraudulent behavior. We plan to model more complicated and elaborate election processes with many more agents and more complex fraudulent behavior. Modeling collusion among agents and developing processes to defend against such intricate frauds and collusions is an important direction in this work. These technologies seem

to be the basis for a discipline of election process engineering and continuous improvement.

Acknowledgements

Matthew Goetz was very helpful at an early stage of this work. Prof. George Avrunin provided valuable feedback about the work. This research was partially supported by the National Science Foundation under Award Nos. CCR-0204321 and CCR-0205575. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of The National Science Foundation, or the U.S. Government.

References

1. Bannet, J., Price, D.W., Rudys, A., Singer, J., Wallach, D.S.: Hack-a-vote: Security issues with electronic voting systems. Security & Privacy Magazine, IEEE, Vol. 2(1). Jan.-Feb. (2004) 32 – 37
2. Cass, A.G., Lerner, B.S., McCall, E.K., Osterweil, L.J., Sutton Jr., S.M., Wise, A.: Little-JIL/Juliette: A process definition language and interpreter. In: Proc. of the 22nd International Conference on Software Engineering, Limerick, Ireland (2000) 754-757
3. Clarke L.A., Chen Y., Avrunin G.S., Chen B., Cobleigh R., Frederick K., Henneman E.A., Osterweil L.J.: Process Programming to Support Medical Safety: A Case Study on Blood Transfusion. In: Proceedings of the Software Process Workshop (SPW2005), Beijing, China, May 25-27 (2005), Springer-Verlag Lecture Notes in Computer Science, Vol. 3840, 347-359
4. Dugger, R.: Counting votes. Annals of Democracy. New Yorker Vol. 64 (38) Nov 7, 1988
5. Dwyer, M.B., Clarke L.A., Cobleigh J.M., and Naumovich G.: Flow Analysis for Verifying Properties of Concurrent Software Systems. ACM Transactions on Software Engineering and Methodology, October (2004) 359-430
6. Kohno, T., Stubblefield, A., Rubin, A., and Wallach, D.: Analysis of an Electronic Voting. System IEEE Symposium on Security and Privacy 2004. IEEE Computer Society Press, May 2004.
7. Robinson S.: Did Your Vote Count? New Coded Ballots May Prove It Did. New York Times, March 2nd 2004, http://www.nytimes.com/2004/03/02/science/02VOTE.html?ex=1084334400&en=88f5c6e6696ccdcf&ei=5070
8. U.S. presidential election, 2000: Wikipedia, http://en.wikipedia.org/wiki/U.S._presidential_election,_2000.
9. Verified Voting Foundation, http://www.verifiedvotingfoundation.org
10. Wise A.: Little-JIL 1.0 language report. Technical Report No. UM-CS-1998-024, Department of Computer Science, University of Massachusetts, Amherst, MA (1998)