This article has been accepted for publication in a future issue of a journal/proceedings, but has not been fully edited. Content may change prior to final publication. Citation information: Belli, F., Hollmann, A., Nissanke, N., Modeling, Analysis and Testing of Safety Issues - An Event-Based Approach and Case Study, Proc. 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Lecture Notes in Computer Science, Vol. 4680, Springer, pp. 276-282

Modeling, Analysis and Testing of Safety Issues - An Event-Based Approach and Case Study

Fevzi Belli[1], Axel Hollmann[1], and Nimal Nissanke[2]

[1] University of Paderborn, Germany
[2] London South Bank University, London, UK

Abstract. This paper proposes an event-based approach with an intuitive simple graphical representation of the system and its environment for designing, analysis and testing safety-critical systems. The events are user actions and system responses, and are ordered according to the threats posed by the resulting system states. This ordering is an integral aspect of the graphical representation, making it possible to directly identify the risks associated with each and every functionally desirable, and undesirable, event relative to one another. Tests that target safety requirements are devised by examining possible traces of these events, represented compactly by regular expressions, exhibiting particular risk patterns such as human error and system failures.

Keywords: Safety, Analysis and Testing, Event Sequence Graphs, Risk Graphs, Regular Expressions, User Interactions.

1 Introduction, Related Work

Modern safety critical systems with automated monitoring and control, sophisticated man-machine interfaces place much greater demands on software to deliver timely response to failures, correct performance of complex tasks and required level of reliability. These are complex requirements vital to assuring system safety [7, 12]. While enhancing the reliability of safety critical systems, man-machine interfaces are known to introduce additional vulnerabilities, requiring careful consideration of various factors such as aspects of automation, psychological issues and ergonomics.

In the face of such vulnerabilities, analysis and testing form an important part of the system development process in revealing and eliminating system's faults. Because of the substantial costs involved in testing, particularly in critical applications requiring extensive testing lasting several months, both testability and the choice of tests to be conducted become an important design consideration. Because of the conflicting demands of minimizing the extent of tests and maximizing the coverage of faults, it is therefore critically important to follow a systematic approach to identifying the test sets that focus on safety, as well as tests that address specific safety requirements.

State-based and event-based methods have been used for almost four decades for specification and testing of software and system behavior, e.g., for conformance testing [3], as well as for specification and testing of system behavior [4] and more recently by [10]. A different approach for testing [8] deploys knowledge engineering methods to generate test cases, test oracles, etc.

F. Saglietti and N. Oster (Eds.): SAFECOMP 2007, LNCS 4680, pp. 276–282, 2007.

Our approach is black-box-oriented. Tests that target safety requirements are devised by examining possible traces of events exhibiting particular risk patterns. These sequences are represented by regular expressions, the undesirable events in them representing human error and system failures, while the desirable events including, in addition to functional ones, various recovery measures to be undertaken following undesirable events. The approach can be used not only for requirements analysis and validation before implementation, but also for analysis and testing of existing code, detecting output faults, detecting erroneous internal states, etc., at low levels of abstraction. This paper is an introduction to our approach and unlike our prior work [2] focuses on safety and testability and demonstrates its use in designing tests that target safety aspects as part of an integrated system development process.

2 Event-Based Modeling of System Vulnerabilities

This work uses *Event Sequence Graphs (ESG)* [2] for representing the user and system behavior. Due to the lack of space, we will give only a brief introduction. Abstractly, an ESG is a digraph $M = (\alpha, E)$, α being a set of nodes uniquely labeled by some *input symbols*, also denoted here by α, and E a non-empty relation on α, with elements in E representing directed arcs between the nodes in α. Each arc represents a pair of consecutive legal events called a *legal event pair (EP)*. *Faulty event pairs* (*FEP*) are the edges of the corresponding $\overline{ESG}$. The set α can be partitioned into two disjoint subsets α_{env} (*environmental* events, e.g., user inputs) and α_{sys} (*system* signals). This distinction is important because events in the latter are controllable within the system, whereas events in the former are not subject to such control.

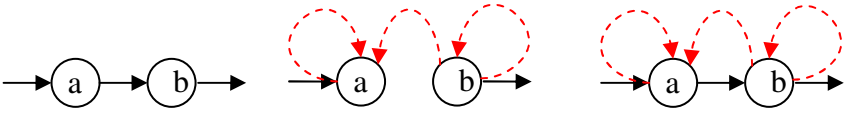

Fig. 1. Example of an ESG, its complement $\overline{ESG}$ and CESG (Completed ESG)

Regular expressions based on alphabet α are used for describing patterns of interactivity between the system and its environment. System *functions* (F), as well as the threats to safety, may each be described using two disjoint subsets of strings, one belonging to the language $L(M)$ and another not belonging to $L(M)$, respectively. *Legal* state transitions are brought about by *desirable* events, leading to symbol sequences belonging to $L(M)$. *Illegal* transitions represent *undesirable* events, leading to faulty symbol sequences not belonging to $L(M)$, signifying breaches to *vulnerabilities* (V).

$$F \subseteq L(M) \text{ and } V \subseteq \overline{L(M)} \tag{1}$$

Risks to safety are often related to the system state. Our approach refers to states indirectly in terms of strings in $L(M)$. Thus, a string $s \in L(M)$ may also be treated as an interchangeable notation for the state reached by the execution of the events in s. The remainder of the vulnerability specification consists of a *risk ordering relation*

$\sqsubseteq$ - a relation on $L(M)xL(M)$. It is defined such that, given states $s_1, s_2 \in L(M)$, $s_1 \sqsubseteq s_2$ is true if and only if the risk level of s_1 to breaches of safety is known to be less than, or equal to, the risk level of s_2 [9]. In this context, *risk level* quantifies the *"degree of the undesirability"* of an event from the safety perspective. A specific benefit of risk ordering in our framework is that it allows a more systematic approach to selection of test cases by focusing on particular vulnerability attributes. The risk ordering relation is intended as a guide to decision making upon the detection of a threat and on how to react to it. The required response to breaches of vulnerability needs to be specified in a *defense matrix* $D \in L(M)xV \rightarrow L(M)$, which is a partial function. The defense matrix utilizes the risk ordering relation to revert the system state from its current one to a less, or the least, risky state. In this sense, D is subject to the following constraint:

$$\forall\, s_1, s_2, v\colon (s_1,v) \in D \wedge D(s_1,v) = s_2 \Rightarrow s_2 \sqsubseteq s_1 \tag{2}$$

This expresses the requirement that, should it encounter the vulnerability v in any given state s_1, the system must be brought down to a state s_2, which is of a lower risk level than s_1. The means by which this is brought about is called an *exception handler*, or a defensive action, which is an appropriately enforced sequence of events. The actual definition of the defense matrix and the appropriate set X of exception handlers is the responsibility of a domain expert specializing in the risks to a given vulnerability. If x is a defense action, then $s_1 x = s_2$. Finally, the model of an application *defended* against vulnerabilities is defined as $M_d = (\alpha, E, F, V, \sqsubseteq, D, X)$.

3 Safety Critical Features and a Case Study

The system under test is a terminal which controls a marginal strip mower (Fig. 2, left top), a vehicle that takes optimum advantage of mowing around guide poles, road signs and trees, etc. The buttons on the control desk (Fig. 2, left bottom) simplify the operation, so that, e.g., the mow head (with revolving knives) returns to working or to transport position when a button is pressed. Beside the positioning, the inclination and the support pressure of the mowing unit can be controlled.

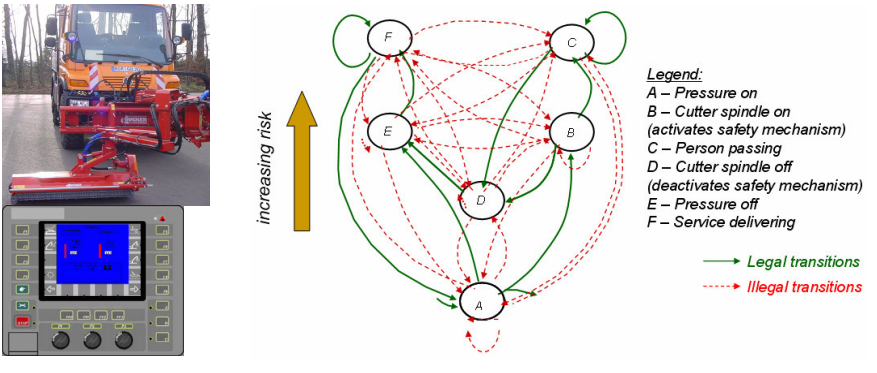

Fig. 2. Marginal strip mower, control desk and its completed ESG model

The ESG in Fig. 2, right, represents the control desk. The set of events α is given by $\alpha_{sys} = \{A,B,D,E\}$ and $\alpha_{env} = \{C,F\}$. Here, the symbol C stands for the event of a passing object, e.g., a person, whereas symbol F denotes a service delivery, e.g., a mechanic repairing the mower unit. The pressure of the mowing unit can be switched on and off and the two events are denoted by symbols A and E. The cutter spindle of the revolving knives is controlled by symbols B and D. When the cutter spindle is switched on (B), a safety mechanism is activated. This mechanism recognizes objects which approach to the mower unit. When the spindle is switched off (D), the safety mechanism is also deactivated. These events bring about hazardous states posing different risks. For example, state C (a person passing) represents a state with the highest risk. As is shown by directed arcs in Fig. 3, the EPs in this example are

$$AE,\ EF,\ FF,\ FA,\ AB,\ BC,\ BD,\ CC,\ CD,\ DE \tag{3}$$

while the complete event sequences (CESs) in any complete cycle of system operation can be represented by the regular expression

$$RegEx = (AEF^{+})^{*}A+((AEF^{+})^{*}ABC^{*}DEF^{+})^{*}A = ((AEF^{+})^{*}\ (\lambda + ABC^{*}DEF^{+}))^{*}A \tag{4}$$

The FEPs shown as dashed lines in Fig. 2 are given by

$$AA,\ DD,\ BB,\ EE,\ AD,\ AC,\ AF,\ DA,\ DB,\ DC,\ DF,\ BA,\ BF,\ BE,\ CA,\ CB,\ CF,\ CE,\ FD,\ FB,\ FC,\ FE,\ ED,\ EB,\ EA,\ EC \tag{5}$$

Expression (4) constitutes *system function F*, while those of (5) the *vulnerability threats V* posed at the junctures corresponding to any matching sub-expression among the ESs that can be generated from (4), e.g., $(AEF^{+})^{*}A$. Each FEP in (5) represents the leading pair of events of an emerging faulty behavioral pattern, with the first event being an acceptable one and the second an unacceptable one. Should the first event of any of the FEPs, e.g., AC, thus happen to match the last event in any of the ESs that can be generated by such a sub-expression, e.g., $(AEF^{+})^{*}A$, then the corresponding pair of ES and FEP, e.g., $(AEF^{+})^{*}\ AC$, describes, or signifies, the occurrence of a specific form of a faulty behavioral pattern.

Table 1. Mower vulnerabilities, the level of the threats posed, and possible defense action

ES (Col. 1)	FEP (Col. 2)	Interpretation (Col. 3)	Comment (Col. 4)	Defense action (Col. 5)
$(AEF^{+})^{*}AB$	BA	Pressure switched on, though already on.	Ignored	–
	BF	Delivering a service, e.g., repair at running system.	Danger	BD
	BE	Switches of pressure though cutter spindle active.	Danger	BD

Table 1 presents an extract of the vulnerabilities relevant to the model given in Fig. 2. In order to overcome the inadequacy of the representation in Table 1, a *risk graph* of the form in Fig. 3 may be used to express the relative risk levels of states with a greater

degree of formality and precision. Each node in the risk graph represents a state described by a regular expression. A directed arc running from a state s_1 to s_2 in Fig. 3 is equivalent to $s_1 \sqsubseteq s_2$, signifying that the risks posed by state s_2 is known to be at the same level as, or exceed, the risks by s_1. In the diagram, an upward running arc signifies that the state lying above poses a greater risk than the one lying below. Arcs drawn in solid lines, as well as the states denoted by underscored ES (regular expressions), refer to the normal functional behavior, while those in dashed lines and other (non-underscored) FES (regular expressions) refer to vulnerability states. For reducing clutter, loops at nodes have been omitted in the diagram.

Having identified potential vulnerabilities of safety concerns, it is possible to provide measures that counteract them. This was what is intended by the concept of *defense matrix* and *exception handlers*. An attempt has been made in Table 1 to propose the defensive actions that may be taken, although due to the limited scope of our model these actions address the potential vulnerabilities only partially.

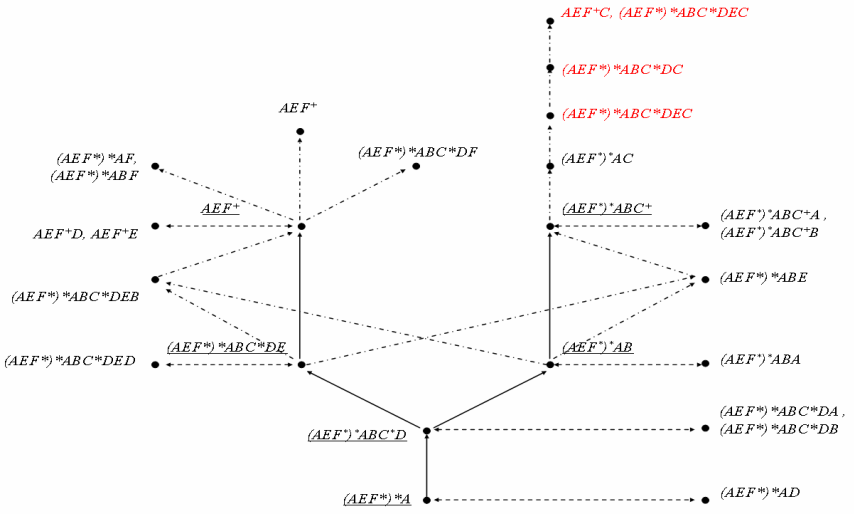

Fig. 3. Risk graph

Should Table 1 have been complete in these respects, the event sequences listed in column 5 would have been equivalent the set of the exception handlers X, while columns 1,2 and 5 would have amounted to a definition of the required defense matrix implicitly, provided that the data in these columns satisfies condition (2). Note that the concatenation of expressions for the current state, columns 1, 2 and 5 in the appropriate manner (by dropping common signals as appropriate) gives the state aimed at by the defense matrix as a result of invoking the corresponding exception handler.

We can use the CESG in Fig. 3 to simulate all potential test scenarios. All what is required is the information given in EPs (3) and FEPs (5), which can be generated by

Table 2. Excerpt of issues detected

No.	Faulty Behavioral Pattern Detected
1	When the function *PressureOn* is deactivated, the function *CutterSpindleOn* can only then be activated if the function *PressureOn* is activated (contradiction!).
2	A change from the view *RSM_Mode_I* to the view *RSM_TranspWorkpos* while function *PressureOn* is active is possible only if the function *PressureOn* is deactivated.
3	When the function *CutterSpindleOn* is activated, the function *PressureOn* can only be deactivated if the function *CutterSpindleOn* is deactivated.

expanding the RegEx (4), applying wellknown algorithms [11]. An excerpt of faulty behavioral patterns detected by the tests is given in Table 2.

4 Conclusion and Future Work

Based on [1, 5, 9], this paper has introduced an integrated design approach capable of addressing system design against system vulnerabilities threatening safety. It allows the consideration of further vulnerabilities threatening other system attributes, such as security and usability, in a single framework and in a similar manner. Incorporation of both the desired and the undesired features of the system in the model allows a practical way to realize the "design for testability" in software design. The degree of undesirability is represented in the form of a risk ordering relation – an expression of relative levels of risks posed by hazardous states. This allows targeting the design of tests at specific system attributes. The framework is based on the concept of 'event sequence graphs'. This could form the basis for the adoption of the approach in other software modeling approaches and tools such as Statecharts [6].

References

[1] Belli, F., Grosspietsch, K.-E.: Specification of Fault-Tolerant System Issues by Predicate/Transition Nets and Regular Expressions – Approach and Case Study. IEEE Trans. On Softw. Eng. 17/6, 513–526 (1991)
[2] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces. In: Proc. 12th Internat'l. Symp. Software Reliability Engineering, pp. 34–43 (2001)
[3] Bochmann, G.V., Petrenko, A.: Protocol Testing: Review of Methods and Relevance for Software Testing. Softw. Eng. Notes, ACM SIGSOFT, 109–124 (1994)
[4] Chow, T.S.: Testing Software Designed Modeled by Finite-State Machines. IEEE Trans. Softw. Eng. 4, 178–187 (1978)
[5] Eggers, B., Belli, F.: A Theory on Analysis and Construction of Fault-Tolerant Systems (in German). In: Informatik-Fachberichte 84, pp. 139–149. Springer, Berlin (1984)
[6] Harel, D., Namaad, A.: The STATEMATE Semantics of Statecharts. ACM Trans. Softw. Eng. Meth. 5, 293–333 (1996)
[7] Leveson, N.G.: Safeware, System Safety and Computers. Addison-Wesley, Reading (1995)
[8] Memon, A.M., Pollack, M.E., Soffa, M.L.: Automated Test Oracles for GUIs. In: SIGSOFT 2000, pp. 30–39 (2000)

[9] Nissanke, N., Dammag, H.: Design for Safety in Safecharts With Risk Ordering of States. Safety Science 40, 753–763 (2002)
[10] Offutt, J., Shaoying, L., Abdurazik, A., Ammann, P.: Generating Test Data From State-Based Specifications. The Journal of STVR 13(1), 25–53 (2003)
[11] Salomaa, A.: Theory of Automata. Pergamon Press, Oxford (1969)
[12] Storey, N.: Safety-critical computer systems. Addison-Wesley, Reading (1996)