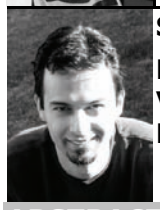
Using Formal Methods to Predict Human Error and System Failures

Bolton, Matthew L.

Systems and Information Engineering / University of Virginia / 151 Engineer's Way / Charlottesville / VA 22904 USA
E-mail: mlb4b@virginia.edu

Bass, Ellen J.

Systems and Information Engineering / University of Virginia / 151 Engineer's Way / Charlottesville / VA 22904 USA
E-mail: ejb4n@virginia.edu



Siminiceanu, Radu I.

National Institute of Aerospace / 100 Exploration Way / Hampton / VA 23666 USA
E-mail: radu@nianet.org

ABSTRACT

Many modern systems are complex in that they depend on the interaction between technical infrastructure (mechanical systems, electrical systems, transportation systems, human-system interfaces, etc.), people (operators, maintenance crews, etc.), and environment conditions to operate successfully. While engineering these subsystems/components, system failures are often emergent as they occur as a result of subsystem interactions. While formal methods, and particularly model checking programs, have proven useful in predicting system failure in computer hardware and software systems, they have not been extensively used to evaluate one of the largest sources of emergent failure, human error; the error resulting from the interaction between human operators and the system. This paper proposes a framework to fill this gap by using formal models of systems, their human-system interfaces, and their operators' normative behaviors in order to predict when erroneous human behavior can occur and how this behavior can contribute to system failure. This framework is illustrated using a model of the Therac 25.

Keywords

Formal methods, human error, system failure, human performance modeling.

INTRODUCTION

With multiple interacting sub-systems and people (operators, maintenance crews, etc.) working towards multiple and sometimes conflicting goals, modern, safety-critical systems are inherently complex. While the majority of the sub-systems (including the interfaces humans use to interact with the system) are well engineered, system failures still occur: airplane crashes, air-traffic conflicts, power plant failures, defense system false alarms, etc. [19]. Such failures are often due not to the breakdown of a single component, but a series of minor events that occur at separate times, ultimately leading to dangerous outcomes. Further, the majority of the mistakes or failures that lead to such outcomes are often the result of human behavior rather than equipment or component failure.

Such failures are difficult to predict as they are emergent features of the complex interactions. Because failures may occur under unexpected and infrequent combinations of conditions, they may not be uncovered during system tests and evaluations. Formal modeling techniques and model checking technology that exhaustively searches the system's operational state space may identify human and sub-system interactions that result in unsafe operating conditions. Using formal models of systems and human behavior with model checkers, this paper proposes a framework for predicting when human error may contribute to system failure.

System Failure

The notion of system failures was popularized by Perrow [19] who referred to system failures as "normal accidents". Perrow's theory states that minor failures will occur in the subsystems of a complex system. While the system may have safeguards to prevent any one failure from drastically impacting system performance, a system accident occurs when multiple failures occur and produce an unanticipated interaction effect.

Reason [21] extended these ideas by identifying two categories of failures that can contribute to system failure: a) *active failure* are those that immediately lead to adverse consequences and b) *latent failures* are those for which the damaging results may not become apparent until much later. Latent failures play an important role in complex system failure as they may lie dormant in the system until they interact with other failures (active or latent) and result in system failure.

Reason noted that the majority of complex systems failures were a particular type of latent failure: human error (a planned sequence of activities that does not produce the intended result during human-system interaction). Reason supported this claim by citing statistics from the Institute of Nuclear Power Operations of 180 incident reports from 1983 and 1984 which indicated that 92% of the aggregate incident root causes were due to human error. Thus, mitigating human error could avert system failure.

Formal Methods and Model Checking

Formal methods is concerned with modeling, specifying, and verifying systems using rigorous mathematical techniques. Model checking is a highly-automated, push-button approach used to verify that a model of a system, usually a finite-state machine, satisfies a set of desired properties, written in some temporal logic, such as CTL (Computational Tree Logic), LTL [5]. Verification is the process of proving that the system meets correctness criteria. For model checking, this is done by exhaustively searching a system's state space in order to determine if these necessary criteria hold. While other forms of verification exist (simulation, experimentation with the actual system, deductive verification using proof techniques), model checking has several advantages: it is automatic, it is decidable, and it can be performed using computationally efficient algorithms.

The model checking approach to verification requires a system model, a system specification, and a verification process (Figure 1) [5]. The modeling task involves converting a system's design into a formal description that can be used

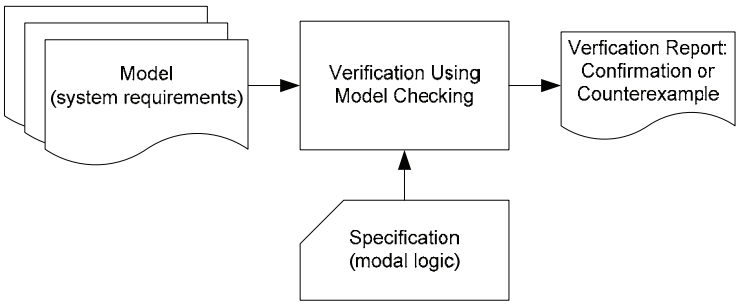


Figure 1. The model checking approach to verification.

by the model checking tool. The specification task involves identifying and defining necessary system properties in a temporal logic (see [20]). The verification task is automatically completed by the model checker itself, with the process producing a report either confirming that the model satisfied the specification, or providing an error trace to illustrate a condition that violated the specification (a counterexample). A failure to verify the model may indicate that there is an error in the model or in the specification which would need to be revised.

The verification process of model checking can combat system failure by finding execution conditions that violate system safety properties. Unfortunately, limited effort has gone into incorporating the single largest source of system failure, human error.

Human Error and Formal Models of Human-System Interaction

Two of the most notable taxonomies for classifying and modeling human error are Reason's Generic Error Modeling System (GEMS) [21] and Hollnagel's phenotypes of erroneous actions [10]. In GEMS, Reason distinguished between three error types based on cognitive factors [21]: slips (and lapses), where the human intends to perform the correct action, but performs the wrong action; rule-based mistakes, where the human performs the wrong action due to an incorrect plan; and knowledge-based mistakes, where the wrong action is performed based on incorrect or incomplete knowledge. Hollnagel's taxonomy describes errors based on their observable forms, with all errors composed of one or more of the following [10]: premature start of an action, delayed start of an action, premature finishing of an action, delayed finishing of an action, omitting an action, skipping an action, re-performing a previously performed action, repeating an action, and performing an unplanned action (an intrusion).

A number of human-system interaction modeling techniques exist including Goals, Operators, Methods, and Selection rules (GOMS) [12], ConcurTaskTrees (CTT) [17], and the Operator Function Model (OFM) [24]. While each has proven useful in modeling normative human behavior, none have explicitly incorporated error taxonomies to model erroneous human behavior. Bastide and Basnyat [1] proposed a system of design patterns for permuting normative CTTs in locations that a human factors specialist deemed appropriate in order to model erroneous behavior. Johnson and Telford [11] have used formal methods to describe how erroneous human behavior contributes to system failures. Lindsay and Connelly [16] have used cognitive modeling to model some erroneous human behavior in an air traffic control task.

An alternate approach to using formal models to predict human error has been used by Kieras and Polson [13], Degani [6], and Leveson and Palmer [14]. Here finite state models of the environment, human user tasks and their interfaces, and system functionality to identify operating conditions that result in incompatibilities between the operator's mental model and the system model, thus predicting potential operator confusion. These analyses have been automated using model checkers in a variety of studies [4][18][22]. Heymann and Degani [9] have investigated extending these methods in order to automatically generate user interfaces. Similar methods have been used to predict sequences of operator tasks that recover the system from and drive the system into unsafe operating conditions [7].

While each of these methods provides insights into human error, all have limitations. Those using cognitive models risk missing many erroneous behaviors due to the inability of the model to replicate the complexity of human cognition. Similarly, those requiring human factors specialists risk human error compromising the design of the erroneous behavior model. Finally, methods used to predict human confusion ignore errors that could result from other cognitive factors.

Objectives

Hollnagel [10] and Reason [21] indicate that there are limited ways in which human error can manifest itself. Given that a system's human interfaces represent the sole means by which human operators can interact with the system, human error will manifest itself through the human-system interface. As it is possible to formally model human-system interfaces [13][6][14], it is possible to automatically predict how the limited manifestations of human erroneous action will cause deviation from normative operator behavior. Because formal models of normative behavior [12][17][24] must conform to the constraints imposed by the human-system interface, they should be able to accommodate the erroneous behaviors they allow.

This paper proposes a framework for using formal models of systems (including their human interface) and the normative behavior of their operators in order to construct models that incorporate erroneous operator behavior. These models can then be used to predict system failure.

A FRAMEWORK FOR PREDICTING HUMAN ERROR AND SYSTEM FAILURE

The proposed framework (Figure 2) extends the model checking verification process (Figure 1). In this extension, there are three inputs: the formal normative human behavior model, the formal system model (which contains a formal model of the human-system interface), and the specification which contains necessary system properties (assertions that system failure conditions do not occur).

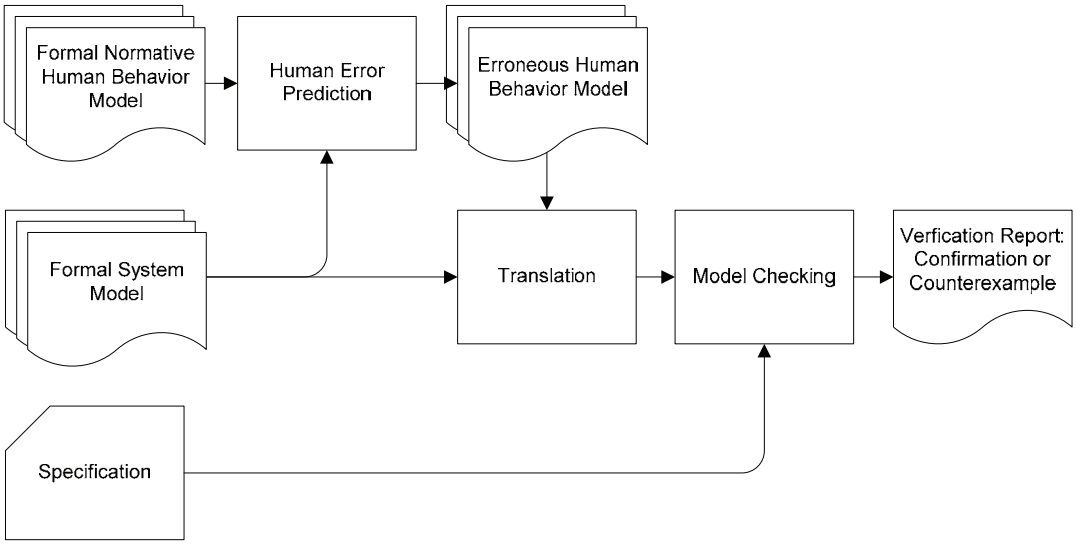


Figure 2. Human error and system failure prediction framework

The framework includes three automatic processes: human error prediction, translation, and model checking. The human error prediction process examines the normative human behavior model and the human-system interface model in order to determine what erroneous human behavior patterns (based on the formal characteristics of human error identified in [10] and [21]) are possible with the interface. It outputs a modified version of the human behavior model with both the normative and erroneous behavior. The translation process takes the system model and the modified human behavior model and converts them into a single model that is readable by the model checker. The model checking process verifies that the system properties from the specification are true in the system model. If verification fails, the process will generate a counterexample showing how the failure condition occurred.

PROOF OF CONCEPT: THE THERAC-25

To illustrate the application of the framework, we consider a simplified version of the Therac-25, a medical system for which human error played an important role in a system failure[1]. The Therac-25 is a room-sized, computer-controlled, medical linear accelerator. It has two modes of operation: the electron beam mode is for shallow tissue treatment (with beam energy between 5 to 25 MeV), and the x-ray mode is for deeper treatments with x-ray photons (25 MeV) [15]. The x-ray mode uses a beam flattener (not used in electron beam mode) to produce a uniform treatment area, which dissipates the treatment. Thus x-ray mode requires an electron beam current approximately 100 times higher than that used for the electron beam mode. An x-ray beam application without the spreader in place can deliver a lethal dose of radiation.

When a patient receives treatment, an administrator positions the patient on a treatment table, manually configures the machine, and leaves the room to administer treatment from a remote computer console. The console allows mode selection (between the x-ray and electron beam modes) using the "x" or "e" keys respectively. The administrator can iteratively approve the elements of the treatment plan by hitting the enter key several times[2]. At any point in this process, the administrator can go back to a previous stage by pressing the up key. Once the treatment plan is confirmed, the administrator can press the "b" key to fire the beam.

While a number of problems have been found with the Therac-25 (see [15]), the most notorious one occurred when a seemingly benign human error resulted in patients being treated with a dangerous dose of radiation. In this error sequence, an operator would intend to apply an electron beam treatment. However, instead of pressing the "e" key, the operator pressed the "x" key, realized the error, pressed the up key to recover from the error, pressed the "e" key, confirmed the treatment plan with multiple presses of the return key, and pressed the "b" key to administer the beam treatment. If this sequence of actions was performed in under 8 seconds, the Therac-25 would apply an x-ray treatment without the spreader in place.

In order to show how the proposed framework (Figure 2) could have predicted this problem, we walk through the application of the framework's steps.

Formal System Model

The first step is to model the system. We present a simplified model of the Therac-25 using the state charts formalism (see [8]). The system is modeled as four concurrent finite state machines (Figure 3): the interface, the beam's power (BeamLevel), the spreader position (Spreader), and the beam's firing state (BeamFire). At initialization, the interface is in the edit mode, the beam is in neither the x-ray or electron beam mode, and the spreader is out of place. If the user presses the "x" or "e" keys (broadcasting SelectX or SelectE commands to the rest of the model), the interface goes into the XRay or EBeam modes respectively. In either mode, the user can confirm a treatment plan by pressing the enter key, fire the beam using the "b" key (broadcasting a Fire command), and confirm treatment by pressing the enter key. In any of the data entry and beam ready states (XDataEntry, XBeamReady, EDataEntry, and EBeamReady), the user can press the up key to return to the previous state.

For the spreader model, the SelectX command moves the spreader in place. The SelectE command moves the spreader out of place.

---

[1] The Therac-25 has been used in other formal methods verification (see [23]), although human erroneous behavior was not modeled.

[2] This particular behavior was different from the original version of the software which required the administrator to enter the treatment plan manually. This new implementation simply copied the treatment data in based on the manual settings made by the administrator during setup.

When the Fire command is seen by the BeamFire model, the machine transitions to the Fired state indicating that the beam has been fired. The machine then automatically transitions (as indicated by the ε input) back to its initial state (Waiting). In doing so, it broadcasts a Reset command to the rest of the model.

When in the initial state (Neither) of the BeamLevel model, the SelectX and SelectE commands set the beam power to the appropriate setting. Attempting to change the power setting once the level has been initially set (XSet or ESet) puts the machine in an intermediate state (XtoE or EtoX) where it takes 8 seconds to transition to the new power setting. A Reset command sets the machine back to the initial state.

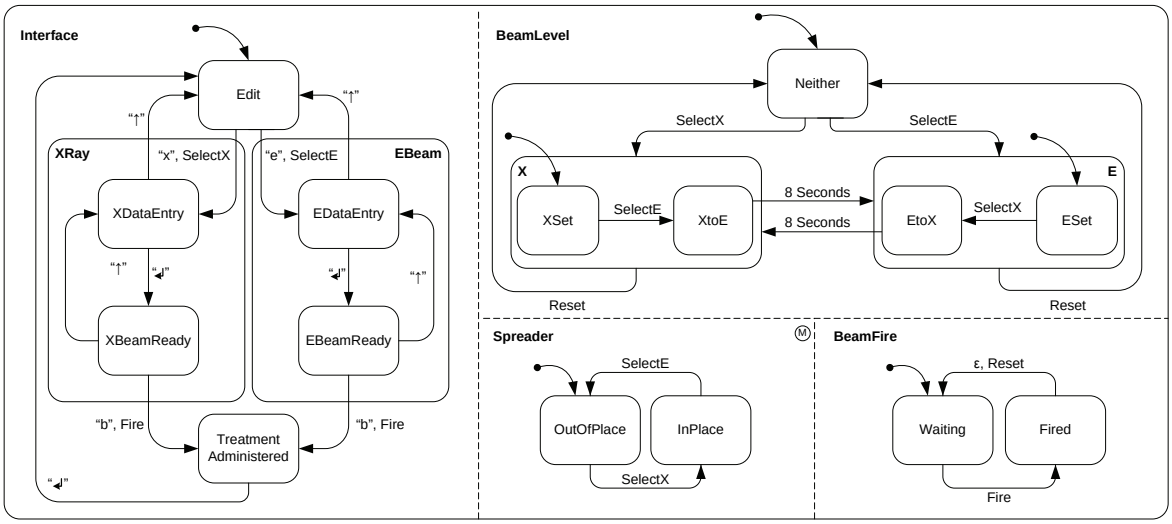


Figure 3. Formal system model of the Therac-25.

Formal Normative Human Behavior Model

We want to simulate erroneous human behavior while applying electron beam treatment. Using the OFM formalism (see [24]), we develop a normative model of human behavior for application of the electron beam treatment (Figure 4). The administration of electron beam treatment activity decomposes into five sequential activities: the selection of the electron beam mode, the confirmation of the treatment plan, the firing of the electron beam, and the confirmation of treatment. Each of these activities maps respectively into an atomic action: pressing the "e" key, pressing the enter key, pressing the "b" key, and pressing the enter key.

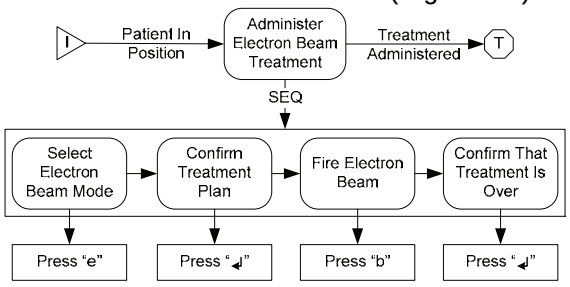


Figure 4. Normative human behavior model for applying electron beam treatment.

Specification

The specification for the system failure condition being evaluated is simply that the beam should never fire at the x-ray level unless the spreader is in place. We can represent this using temporal logic (see [20]) as follows:

$$\mathbf{AG}\ \neg(\text{BeamLevel} = \text{X} \wedge \text{Spreader} = \text{OutOfPlace} \wedge \text{BeamFire} = \text{Fired})$$

This can be translated as: for all paths through the model, there will never be a state where the intensity of the beam is at the x-ray level, the spreading is out of place, and the beam is fired.

Human Error Prediction and Erroneous Human Behavior Specification

With the normative model of human behavior (Figure 4) and a formal model of the human-system interface (Figure 3), one can predict the human error that caused the system failure. When an operator begins to administer electron beam treatment, the interface will be in the Edit state. There are two options for progressing to another state: pressing the “e” key (correct course of action according to the normative human behavior model) and pressing the “x” key. Doing the latter constitutes a common type of erroneous human behavior, that of performing a familiar action at the inappropriate time (Reason’s slip [21] and Hollnagel’s [10] erroneous act). If this latter action occurs, the human will need to recover from the error in order to resume electron beam treatment. The interface indicates that this can occur if the human presses the up arrow. He can then resume electron beam administration as he normally would. Figure 5 illustrates a human behavior model encompassing both the normative and erroneous behavior.

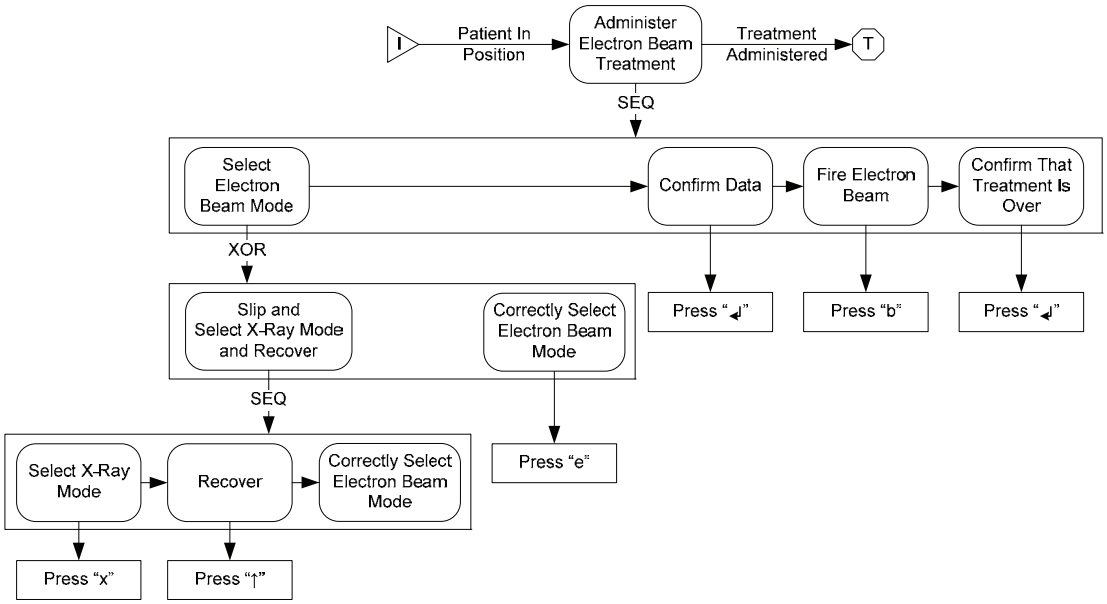


Figure 5. Erroneous human behavior model. Note that in an XOR decomposition only one of the sun-activities is executed.

Translation

The next step is to translate the erroneous human behavior model and the system model into a single model that can be evaluated by a model checker. This aggregate model is presented in the state charts formalism (Figure 6). The only addition is the human behavior model, represented as its own machine. In this translation, the human behavior starts in an initial state and can transition to the erroneous or the normative behavior sequence. In each, states in the model represent intermediary periods between user activities. State transitions represent human activities, with those activities being broadcast to the rest of the model.

Although not explicitly shown here, part of this proof of concept involved manually implementing the model in Figure 6 in the SAL (Symbolic Analysis Laboratory) [2] and SMART (Stochastic Model Checking Analyzer for Reliability and Timing) [3] model checking programs.

Model Checking and Counterexample

When the aggregate system model and specification are run through the model checkers, the specification is violated (the system failure condition occurs). As a result,

the model checkers produce the following state trace (illustrated in Figure 6):

1. The process starts with each machine in its initial state.
2. The machine prepares to execute the erroneous behavior sequence.
3. The human behavior model has pressed the “x” key, the interface is now in data entry mode for the x-ray, the x-ray beam power level is set, and the spreader is put in place.
4. The human has pressed the up key to correct for the erroneous act, the interface is back in its initial state.
5. The human has pressed the “e” key to resume correct electron beam administration, the beam power level is set to transition to the electron beam power setting in 8 seconds, and the spreader is moved out of place.
6. The human confirms treatment data by pressing the enter key.
7. The human has pressed the “b” key, and the beam is fired at the X-ray power level with the spreader out of place, the system failure condition.

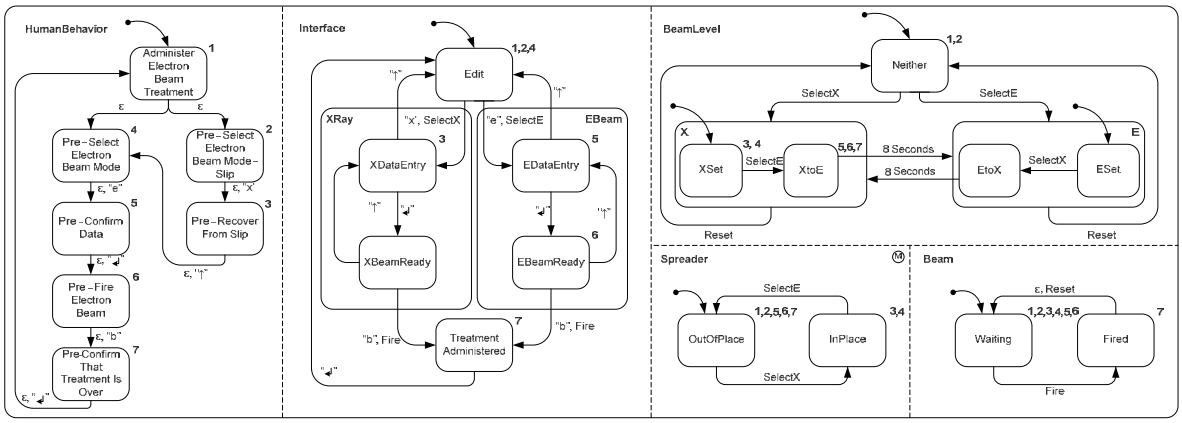


Figure 6. Combined human behavior model and system model with execution trace to indicate failure sequence.

DISCUSSION

This simple example shows that it is possible to use formal models of systems and the normative behavior of their human operators with the proposed framework in order to predict human error and to forecast system failure. However, there are areas that require further development before the framework process is complete.

Human Error Prediction

While the erroneous human behavior was easy to manually predict in this example, future work will make this process automatic and capable of encompassing more types of erroneous behavior. Hollnagel’s [10] error phenotypes are particularly useful in that they are able to model the vast majority of human erroneous behavior based on the combination of timing and plan-divergence errors observable at individual action level. This is particularly convenient in the context of the OFM. Because OFMs decompose activities into their atomic actions, erroneous behavior models can be constructed by replacing each atomic action with all of the possible single action errors that can manifest themselves at that action.

OFM Specification and Translation

There are currently no tools designed to specify OFMs as part of larger formal system

model specification. Such tools will be necessary to ensure that OFMs are compatible with the human interface specifications given that both are dependent on each other. Such compatibility will also be important given that it will be necessary to translate the OFMs into the same notation used for the system model so that they can be processed by the model checker.

Specification

While some undesirable system conditions may be easy to identify, and therefore incorporate into the specification, properties of some emergent failures may be less obvious. It may be useful to develop additional processes to identify failure conditions by examining the system model. Here, Reason's notion of latent errors [21] could be very helpful. In the context of erroneous human behavior, a latent failure would be one in which an erroneous behavior causes the system to transition to a state inconsistent with that associated with the normative human behavior. The Therac-25 example contains such a condition: the human's slip and recovery actions resulted in the system being in a different state than they had been before the erroneous behavior.

Conclusions

While the process depicted in this framework is still under development, the example shows that it is feasible. With additional work, this integrated human-systems modeling approach to system verification will provide significant insight into safety critical systems and the prevention of system failures. Future work will apply these methods to applications in the medical and aerospace domains.

ACKNOWLEDGEMENTS

The project described was supported in part by Grant Number T15LM009462 from the National Library of Medicine and Research Grant Agreement UVA-03-01, sub-award 6073-VA from the National Institute of Aerospace (NIA). The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIA, NASA, the National Library of Medicine, or the National Institutes of Health.

REFERENCES

[1] Bastide, R.; Basnyat, S., 2006, Error patterns: Systematic investigation of deviations in task models, In Coninx, K.; Luyten, K.; Schneider, K., eds. TAMODIA 2006, 109-121.

[2] Bensalem, S.; Ganesh, V.; Lakhnech, Y.; Munoz, C.; Owre, S.; Rues, H.; Rushby, J.; Rusu, V.; Saïdi, H.; Shankar, N.; Singerman, E.; Tiwari, A., 2000, An overview of SAL, In C. M. Holloway, editor, 2000, Fifth NASA Langley Formal Methods Workshop, 187-196.

[3] Ciardo, G.; Jones, R. L; Miner, A. S.; Siminiceanu, R., 2006, Logic and stochastic modeling with SMART, performance evaluation, 63, 578-608.

[4] Crow, J.; Javaux, D.; Rushby, J., 2000, Models and mechanized methods that integrate human factors into automation design, International Conference on Human-Computer Interaction in Aeronautics, Toulouse, France.

[5] Clarke, E.; Grumberg, O.; Peled, D., 1999, Model Checking, MIT Press: Cambridge.

[6] Degani, A., 1996, Modeling human-machine systems: On modes, error, and patterns of interaction, Unpublished doctoral dissertation, Atlanta, GA: Georgia Institute of Technology.

[7] Degani, A.; Heymann, M.; Barshi, I., 2005, A formal methodology, tools, and algorithm for the analysis, verification, and design of emergency procedures and recovery sequences, NASA internal white paper. Mountain View, CA: NASA Ames Research Center.

[8] Harel, D., Statecharts: A visual formalism for complex systems, Science of Computer Programming, 8, 231-274.

[9] Heymann, M.; Degani, A., 2007, Formal analysis and automatic generation of user interfaces: approach, methodology, and an algorithm, Human Factors, 49, 311-330.

[10] Hollnagel, E., 1993, The phenotype of erroneous actions, International Journal of Man-Machine Studies, 39, 1-32.

[11] Johnson, C. W.; Telford, A. J., 1996, Extending the application of formal methods to analyze human error and system failure during accident investigations, Software Engineering Journal, 11, 355–365.

[12] Kieras, D.; John, B., 1994, The GOMS family of analysis techniques: tools for design and evaluation, CMU-HCII-94-106.

[13] Kieras, D. E.; Polson, P. G., 1985, An approach to the formal analysis of user complexity, International Journal of Man-Machine Studies, 22, 365-394.

[14] Leveson, N.; Palmer, E., 1997, Designing automation to reduce operator errors, Proceedings of Systems, Man, and Cybernetics Conference, October.

[15] Leveson, N. C.; Turner, C. S., 1993, An investigation of the therac-25 accidents, IEEE Computer, July, 18-41.

[16] Lindsay, P.; Connelly, S., 2001, Modeling erroneous operator behavior for an air-traffic control task, Australian Computer Society 3rd Australian User Interface Conference, 43-55.

[17] Mori, G.; Paternò, F.; Santoro, C., 2002, CTTE: support for developing and analyzing task models for interactive system design, IEEE Transactions on Software Engineering, 28, 8, 797-813.

[18] Oishi, M., 2004, User-interfaces for hybrid systems: analysis and design through hybrid reachability, Unpublished doctoral dissertation. Stanford, CA. Stanford University.

[19] Perrow, C, 1984, Normal Accidents: Living with High-Risk Technologies, New York: Basic Books.

[20] Pnueli, A., 1977, The temporal logic of programs, *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, 46-67.

[21] Reason, J., 1990, Human Error, Cambridge, England: Cambridge University Press.

[22] Rushby, J., 1999, Using model checking to help discover mode confusions and other automation surprises, 3rd Workshop on Human Error, Safety, and System Development. Liege, Belgium, 7-8 June.

[23] Thomas, M., 1994, A proof of incorrectness using the LP theorem prover: The editing problem in the Therac-25, High Integrity Systems Journal, 1, 1, 35-48.

[24] Thurman, D. A.; Chappell, A. R.; Mitchell, C. M., 1998, An enhanced architecture for OFMspert: A domain-independent system for intent inferenceing. IEEE International Conference on Systems Man and Cybernetics.