A Method for the Formal Verification of Human-interactive Systems

Matthew L Bolton
University of Virginia
Systems and Information Engineering
Charlottesville, Virginia

Ellen J. Bass
University of Virginia
Systems and Information Engineering
Charlottesville, Virginia

Predicting failures in complex, human-interactive systems is difficult as they may occur under rare operational conditions and may be influenced by many factors including the system mission, the human operator's behavior, device automation, human-device interfaces, and the operational environment. This paper presents a method that integrates task analytic models of human behavior with formal models and model checking in order to formally verify properties of human-interactive systems. This method is illustrated with a case study: the programming of a patient controlled analgesia pump. Two specifications, one of which produces a counterexample, illustrate the analysis and visualization capabilities of the method.

INTRODUCTION

Complex, safety-critical systems involve the interaction of automated devices and goal-oriented human operators in a dynamic environment. Failures in such systems are often not due to a single component, but rather system components interacting in unexpected ways. One source of such problems is the interaction between the human operator and other system components. For example, human behavior has contributed to 44,000-98,000 deaths nationwide every year in medical practice (Institute of Medicine, 2000), 73.8% of all general aviation accidents (AOPA Air Safety Foundation, 2007), and other high profile disasters (see for example Perrow, 1984).

Formal methods are mathematically based languages, techniques, and tools for specifying and verifying systems (Clarke & Wing, 1996). Model checking is an automated approach to verify that a model of a system, usually a finite-state machine, satisfies a set of desired properties (a specification) written in a temporal logic (Clarke, Grumberg, & Peled, 1999). Verification is the process of proving that the system meets the specification's criteria. For model checking, this is achieved by exhaustively searching a system's state space in order to determine if these criteria hold. If there is a violation, an error trace is produced (a counterexample). This counterexample depicts an incremental sequence of model states (specific valuations of the model's variables) that ultimately lead to a state in which the specification has been violated: an execution trace illustrating how a failure occurred. By examining the counterexample, analysts can determine why the specific verification failed and then develop interventions.

Researchers have leveraged both human factors and formal methods to evaluate safety-critical systems. Such synergy has been used for identifying the cognitive precondition for mode confusion and automation surprise (Crow, Javaux, & Rushby, 2000; Degani, 1996; Javaux, 2002; Rushby, 1999) and the automatic generation of emergency procedures, and recovery sequences (Degani, Heymann, & Barshi, 2005).

The use of task behavior as part of a larger, formal system model is potentially useful for accident/disaster prevention because it allows the ramifications of different human behaviors to be verified in relation to other aspects of the system. Two approaches have been tried: 1) Incorporating cognitive architectures as part of the formal model in order to drive human task models (Curzon, Ruksenas, & Blandford, 2007); and 2) Only modeling observable human task behavior where other model events drive task execution (Fields, 2001).

There are limitations to these approaches. Firstly, the human task models have been constructed using either custom intermediary languages specifically designed to allow the task behavior models to fit within the formal modeling environment (Fields, 2001) or the formal modeling notation itself (Curzon, et al., 2007). These methods do not take advantage of existing task model representations such as hazard networks (Bass, et al., 2004), operator function models (OFMs) (Thurman, Chappell, & Mitchell, 1998), and ConcurrTaskTrees (CTTs) (Patternò, 1999). Similarly, when a specification is violated, model checkers produce a difficult to interpret counterexample. Visual representations of task models could be used to facilitate counterexample evaluation. In addition, current applications have limited scope. Fields (2001) only considered models of human task behavior concurrently with models of device automation. Curzon et al. (2007) utilized models of human cognition, human tasks, the human-device interface (HDI), and the device's automation. As a result, analyses were conducted with different human cognitive and task models (representing different designations of users) and different interfaces. Also, such models have only been used to model passive devices: when behavior is solely driven by its human operator (Blandford, Butterworth, & Curzon, 2003). Because complex system failures can result from the interactions between the environment, the automated device, the HDI, the human operator's goals, and his or her task behavior, a complete, system-level analysis would consider interconnected models for all of these. Thus, while each of the approaches can provide useful insights into human-device interaction, they are not capable of supporting a full system-level analysis. Further, because models of human-interactive systems can quickly become intractable due to the state space explosion (Bolton & Bass, 2009b), the additional infrastructure required to model user cognitive function (as in Curzon, et al., 2007) would significantly limit what can be modeled in the other system elements (the device, the environment, etc.), making it less practical for system-level work.

In order to allow human factors engineers to exploit their existing modeling technologies with the powerful verification capabilities of formal methods, we have developed a method for the formal modeling of human-interactive systems. In order to perform system-level verifications, this method utilizes

a framework containing concurrent models of human operator task behavior, human mission directives, device automation, the HDI, and the operational environment which are composed together to form a larger, formal system model (see Bolton & Bass, 2009b for more details). To avoid modeling limitations associated with the complexities of modeling cognitive function, this method considers only models of observable human task behaviors, where different cognitive assumptions may influence the composition of task models. Finally, this method utilizes existing task model representations to aid in the modeling and analysis process. These representations are used to help visualize counterexample data. This paper outlines this method and illustrates its use with a real world case study: a patient controlled analgesia (PCA) pump.

METHOD

Method Overview

Initially proposed in Bolton, Bass, and Siminiceanu (2008) and updated in Figure 1, a human modeler examines documentation and other information gleaned from evaluation of a target system. He creates models of the human task behavior using a task analytic representation. Our automated process, the task model translator, then converts the human task model into a formal human task model which can be incorporated into the larger formal system model. The modeler creates (or uses existing) formal system models of the mission, HDI, device automation, and environment. He also creates temporal logic specifications representing the system qualities to be verified.

The formal system model and the system specification are run through the model checker which produces a verification report. If a violation of a specification is found, the report will contain a counterexample. The counterexample and the original human task behavior model can then be processed by our automated visualizer which illustrates the sequence of human behaviors and related system states that led to the violation.

Target System

This method was used to evaluate the Baxter Ipump. Modeling information was gleaned from its operator manual (Baxtor, 2006), direct interaction with the device, and interactions with healthcare professionals familiar with it. This PCA pump is an automated machine that allows for the controlled delivery of sedative, analgesic, and anesthetic medication solutions via intravenous, subcutaneous, or epidural routes.

Pump behavior is dictated by internal automation, some of which is dependent on how the pump is programmed. This programming is accomplished via the pump's human-device interface (Figure 2) which contains a dynamic LCD display, and eight buttons. The human operator uses the HDI to program a prescription. This involves the specification of the operational mode, fluid volume of the medication to be delivered, PCA dose, delay between doses, continuous rate of medication delivery, limit on the amount of medication delivered in an hour, and an initial bolus dose. The start and stop buttons start and stop the delivery of medication at certain times during programming. The on-off button is used to turn the device on and off. The LCD display is used to select prescription options (such as operational mode) and specify prescription values. When the operator must choose between two or more options: the interface message indicates what is being chosen and the initial or default option is displayed. The up button is used to scroll through the available options.

When a numerical value is required (such as the volume of a PCA dose), the value's name is listed in the interface message and the displayed value is presented with the cursor under one of the value's digits. The programmer can move the position of the cursor by pressing the left and right buttons. He can press the up button to scroll through the different digit values available at that cursor position. The clear button sets the displayed value to zero. The enter button is used to confirm values and treatment options.

Aside from the administration of treatment, the primary form of the pump's automation is its dynamic checking and restriction of operator entered values. Thus, in addition to having hard limits on the maxima and minima a value can assume, the extrema can change dynamically in response to other user specified values.

Formal Modeling

All of the formal models were constructed using the Symbolic Analysis Laboratory (SAL) language (Moura, Owre, & Shankar, 2003). Formal models of the mission were represented as a set of viable prescription options. The mission (prescription), human task, HDI, and device automation were independently modeled. The behaviors of the automated sys-

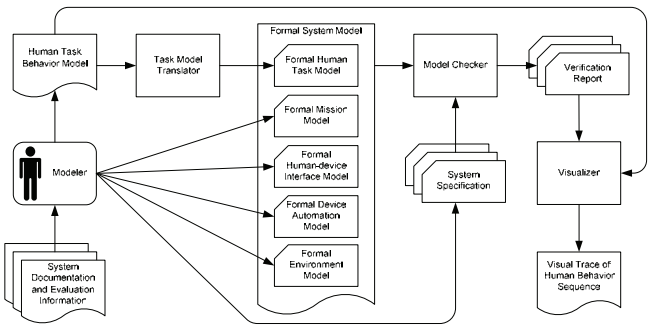

Figure 1. Formal verification methodology for a human-interactive system.

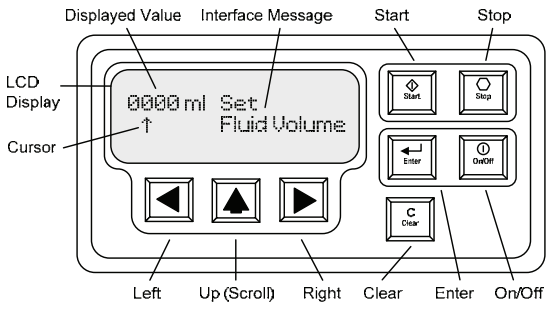

Figure 2. A simplified representation of the Baxter Ipump's human-device interface. Note that the actual pump contains additional controls and information conveyances.

tem and HDI models were kept as representative as possible. Because the PCA pump generally operates in a controlled environment, no environmental model was included. Because of the limited available pump documentation, the system automation model was restricted to pump programming procedures. For more information about this modeling process, model granularity, and the rationale behind the modeling decisions see Bolton and Bass (2009b).

Specifications

All verifications were performed using SAL-SMC, the SAL symbolic model checker. Thus, all specifications were written in linear temporal logic (LTL), the specification logic supported by SAL-SMC. The first specification was intended to verify that the pump would allow the human operator to program any of the mission prescriptions using the human task behaviors from the manual. In LTL this was represented as:

$$\mathbf{AG}((\mathit{InterfaceMessage} = \text{TreatmentAdministering}) \Rightarrow (\mathit{PrescribedTreatment} = \mathit{EnteredTreatment})) \quad (1)$$

Here the *PrescribedTreatment* and *EnteredTreatment* variables denote multiple variables used to describe the prescribed and entered prescriptions respectively. **AG** is a LTL operator which asserts that the subsequent proposition is true along all paths through the model. Thus, Equation 1 can be interpreted as: for all paths through the model, if the interface indicates that treatment is being administered, this should always imply that the prescribed treatment from the mission is equal to the treatment that was entered/programmed into the pump.

Because the specification in Equation 1 was expected to verify (to be proven true), its associated verification was not expected to produce a counterexample. In order to illustrate how counterexample data could be handled using our method, a specification that would not verify was needed. For this, we assume that we never want the device to allow the programmer to enter a prescription of a 9.0 ml/hr continuous medication dose for 100 ml of medication with no bolus.

With *UndesiredPrescription* denoting variables defining this prescription, this specification was given as:

$$\mathbf{AG}\neg(\mathit{InterfaceMessage} = \text{TreatmentAdministering} \wedge \mathit{UndesiredPrescription} = \mathit{EnteredTreatment}) \quad (2)$$

This can be interpreted as: for all paths through the model, the interface will never indicate that treatment is administering with the undesired prescription equaling the entered treatment.

Human Task Modeling and Translation

Human task models were based on the OFM (Thurman, et al., 1998), a task analytic modeling paradigm that specifies the conditions under which operator activity is undertaken. Activities are decomposed into lower-level activities and, finally, actions. Operators on each decomposition specify the temporal relationships between the sub-acts. Actual models were written in an XML-based, generic, OFM-like language called Enhanced Operator Function Model (EOFM) (Bolton & Bass, 2009a) and visualized using our custom visualization tool.

All models were derived from the pump programming task descriptions given in the pump's operator's manual (Baxter, 2006). An example EOFM appears in Figure 3: a task model for programming a prescribed medication fluid volume into the pump. Variables beginning with *i* represent inputs to the human task behavior model (from either the mission or human-device interface models), variables beginning with *a* represent activities, and variables beginning with *h* represent human actions.

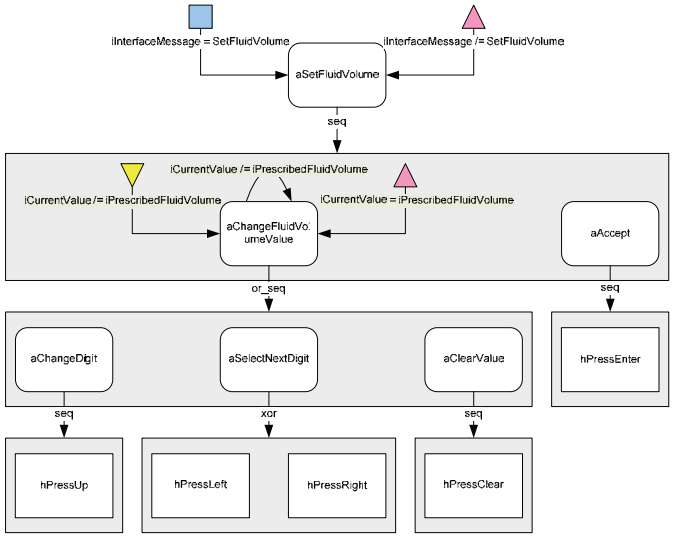

Figure 3. An enhanced operator function model for specifying the fluid volume of the medication in the Baxter Ipump.

As long as the pump's interface is in the "set fluid volume" mode (an invariant condition), the activity for setting the fluid volume can be executed by sequentially performing the activities for changing the fluid volume value and accepting that value. Changing the fluid volume value can commence if the current displayed value does not match the prescribed value (a precondition). As long as this is true (a while condition), the operator should perform one of three activities: changing the digit currently indicated by the cursor; selecting a different digit; or clearing the displayed value. Each of these activities decomposes into its respective actions: pressing the up button; pressing either the left or right buttons; or pressing the clear button. Once the current displayed value matches the prescribed value (a post-condition), the activity for changing fluid volume completes. The activity to accept a displayed value is completed by performing the action for pressing the enter key. The activity for setting the fluid volume ends when the interface is no longer in the "set fluid volume" mode.

The final set of human task models contained twelve EOFMs for performing all of the following tasks: turning the pump on, turning the pump off, starting and stopping medication infusion, setting numerical values for prescriptions (one model for each enterable numerical value), and selecting treatment options (one for each treatment option).

Translation of the EOFM models into SAL code occurred via our java-based software program that recursively parses each EOFM and converts it into SAL code.

Model Checking and Counterexample Visualization

Once the modeling and translation were completed, the formal system model was evaluated by the model checker in order to verify the specifications. When a specification failed to verify, a counterexample was produced. This was concurrently processed with the original EOFM task model specification by our visualizer to illustrate the order in which EOFM activities and actions were executed. In each step from the counterexample, the executing EOFM was displayed with each of the executing activities and/or actions highlighted in green. Activities and actions that had finished executing in prior steps, and were thus not capable of executing in the next step, appeared in gray. When considered concurrently with state information from the counterexample, this allowed analysts to see what human task behavior executed in a particular counterexample step (possibly in response to state information from a previous step), and observe the impact of that behavior on the values of state variables in subsequent steps. Figure 4 illustrates two steps of a counterexample visualization.

VERIFICATION RESULTS

The EOFMs and system models discussed were used in two separate verification procedures. The first, using the specification in Equation 1, was meant to verify that mission prescriptions would always be correctly programmed into the pump using the existing HDI and device automation with the human task behavior specified in the manual. The second, using the specification in Equation 2, was meant to produce a counterexample illustrating a faux-error condition where a specific prescription is programmed.

Verification 1

The specification from Equation 1 was verified as being true, proving that the device automation and HDI was capable of always allowing a human operator to program mission prescriptions using the task behavior described in the pump's manual and training materials. The verification process required a search depth of 212 (an examination of all paths through the model that had up to 212 unique configurations of variables) and took 45 minutes.

Verification 2

Verification of the specification from Equation 2 returned the expected counterexample containing 144 steps (the depth of the search) after approximately 50 minutes. When run through the visualizer, this revealed a human behavior sequence containing the execution of 22 user actions in which the operator programmed a prescription of a 9.0 ml/hr continuous medication dose for a bag containing 100 ml of medication with no bolus. A subsection of this visualization (steps 35 and 36) appears in Figure 4. The full visualization can be seen at http://cog.sys.virginia.edu/formalmethods/ce/. Thus, if Equation 2 had represented a true safety critical property, this counterexample visualization would allow the analyst to see exactly how human task behavior helped drive the system into a state where this property is violated.

DISCUSSION

This paper shows that it is possible to perform a verification of a human-interactive system using the method depicted in Figure 1: a process capable of both verifying a specification and providing a counterexample when a specification is violated. Thus, it is possible to perform a system-level verification analysis of a human-interactive system using task analytic models to both model human task behavior and represent counterexamples.

Because there may be more than one use for counterexample visualizations (model debugging and failure sequence comprehension are two possibilities), there may be different requirements for each. Future work will investigate what the potential uses are for counterexample visualization and to re-

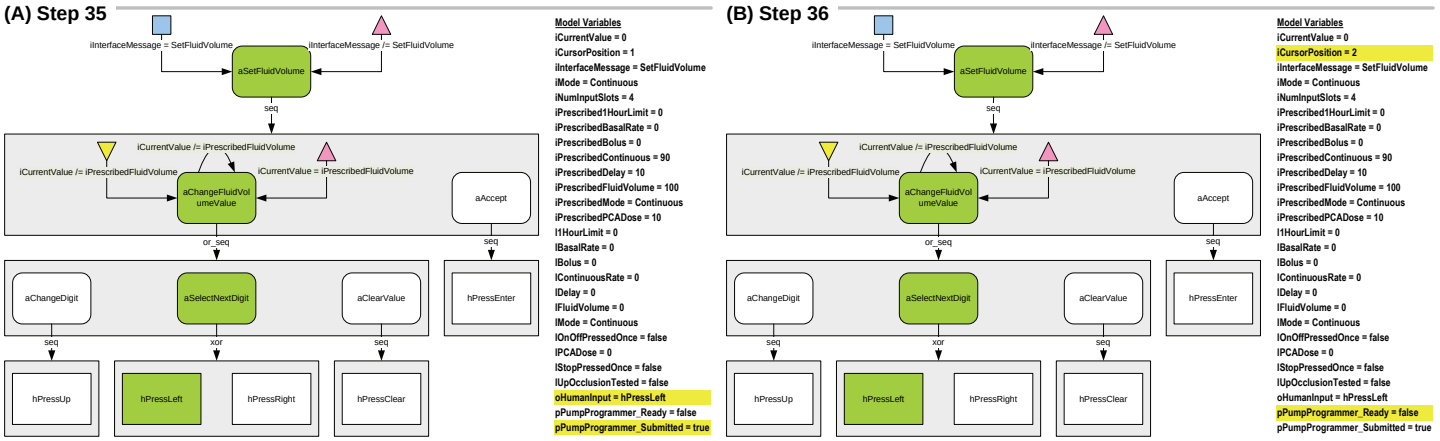

Figure 4. Two steps from a counterexample visualization, depicting the relevant human task (EOFM) visualization concurrently with variable valuations of non-human task related model variables. Active human task elements are highlighted in green. Model variables whose values have changed since the previous step are highlighted in yellow. (A) The user has pressed the left button and indicated that input has been submitted by setting *pPumpProgrammer_Submitted* to true. (B) The interface has received the input (as indicated by *pPumpProgrammer_Ready* becoming false) and the position of the cursor (*iCursorPosition*) has updated in response.

fine the requirements for each.

This paper focused on predicted, normative human behavior models. However, it could also be used to evaluate human behavior models derived from a more observational cognitive task analyses. Similarly, while it is possible for normative human behavior to contribute to system failure, system failures are most commonly associated erroneously human behavior. The method presented in this paper could potentially be used to assess the impact of erroneous human behavior on a system's ability to satisfy specifications. In its current form, this would require the analyst to manually incorporate erroneous behaviors into the EOFMs (see Fields, 2001; Bolton, Bass, & Siminiceanu, 2008). However, because erroneous human deviations from task behavior occur in a very systematic ways (Hollnagel, 1993b), and because human behavior is inherently limited by the human-device interface, it should be possible to automatically incorporate erroneous behaviors into normative EOFMs by examining the formal model of the HDI concurrently with normative EOFMs. Future work will investigate incorporating this capability into the method. This would allow the impact of different types of erroneous human behavior to be deterministically proved rather than stochastically assessed using traditional human reliability analysis (Hollnagel, 1993a).

Finally, the method described here is limited by the current capabilities of model checking: where limitations of modern computation and model checking technology impact what systems can be modeled and at what level of detail (Clarke, Grumber, & Peled, 1999). Compromises to model scope and representativeness were made in order to produce a model that was capable of being verified (see Bolton & Bass, 2009b). Future work will seek to characterize the limitations of the formal modeling and verification of human interactive systems.

ACKNOWLEDGEMENT

The project described was supported in part by Grant Number T15LM009462 from the National Library of Medicine (NLM) and Research Grant Agreement UVA-03-01, subaward 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 NLM, or the National Institutes of Health.

The authors would like to thank Radu I. Siminiceanu, of the NIA and Ben Di Vito of NASA Langley for their help with the technical issues in this paper. They would like to thank Diane Haddon, John Knapp, Paul Merrel, Kathryn McGough, and Sherry Wood of the University of Virginia Health System for helping them understand the functionality of the Baxter Ipump and providing the necessary documentation, training materials, and device access.

REFERENCES

AOPA Air Safety Foundation (2007). *2007 Nall Report: Accident Trends and Factors for 2006*. AOPA Air Safety Foundation. Available at http://www.aopa.org/asf/publications/07nall.pdf

Bass, E.J., Ernst-Fortin, S.T., Small, R.L., & Hogans, Jr., J.T. (2004). Architecture and development environment of a knowledge-based monitor that facilitate incremental knowledge-base development. *IEEE Transaction on Systems, Man, and Cybernetics, Part A, 34*(4), 441-449.

Baxter Heath Care Corporation. (2006). *Ipump pain management system operator's manual*. Baxter Heath Care Corporation.

Blandford, A., Butterworth, R., and Curzon, P. (2004) Models of interactive systems: a case study on programmable user modeling. *International Journal of Human-computer Studies, 60*(2), pp. 149–200.

Bolton, M. L. & Bass, E. J. (2009a). Enhanced operator function model: A generic human task behavior modeling language. *IEEE International Conference on Systems Man and Cybernetics*, October 11-14, 2009, San Antonio, Texas.

Bolton, M. L. & Bass, E. J. (2009b). Building a formal model of a human-interactive system: Insights into the integration of formal methods and human factors engineering. *First NASA Formal Methods Symposium*, April 6 - 8 Moffet Field, CA.

Bolton, M. L., Bass, E. J., Siminiceanu, R. I. (2008). Using formal methods to predict human error and system failures. *2008 Applied Human Factors and Ergonomics International Conference*, July 14-17, 2008, Las Vegas, NV.

Clarke, E. M., Grumberg, O., & Peled, D. (1999). *Model Checking*. MIT Press: Cambridge.

Clarke, E. M., & Wing, J.M. (1996). Formal methods: State of the art and future trends. *ACM Computing Surveys, 28*(4):626--643.

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.

Curzon, P., Ruksenas, R. & Blandford, A. (2007) *An approach to formal verification of human-computer interaction*. Formal Aspects of Computing. 513-550. DOI 10.10.1007/s00165-007-0035-6.

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

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.

Fields, R. E. (2001) Analysis of erroneous actions in the design of critical systems. D. Phil Thesis, Technical Report YCST 20001/09, University of York, Department of Computer Science.

Hollnagel, E. (1993a). *Human reliability analysis: Context and control*. Academic Press.

Hollnagel, E.. (1993b). The phenotype of erroneous actions. *International Journal of Man-Machine Studies, 39*, 1-32.

Institute of Medicine. (2000). To Err Is Human: Building a Safer Health System. *The National Academies Press*.

Javaux, D. (2002). A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system. Reliability Engineering & System Safety, 75, 147-165.

Moura, L., Owre, S., & Shankar, N., (2003). The SAL Language Manual. CSL Technical Report SRI-CSL-01-02 (Rev. 2).

Paternò, F. (1999). *Model-based Design and Evaluation of Interactive Applications.* Springer Verlag.

Perrow. C. (1984). *Normal Accidents: Living with High-Risk Technologies.* New York: Basic Books.

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.

Thurman, D. A., Chappell, A. R., Mitchell, C. M. (1998). An enhanced architecture for OFMspert: A domain-independent system for intent inferencing. *IEEE International Conference on Systems Man and Cybernetics.*