Building a Formal Model of a Human-interactive System: Insights into the Integration of Formal Methods and Human Factors Engineering

Matthew L. Bolton *
University of Virginia
Charlottesville, VA, United States of America
mlb4b@virginia.edu

Ellen J. Bass†
University of Virginia
Charlottesville, VA, United States of America
ejb4n@virginia.edu

Abstract

Both the human factors engineering (HFE) and formal methods communities are concerned with finding and eliminating problems with safety-critical systems. This work discusses a modeling effort that leveraged methods from both fields to use model checking with HFE practices to perform formal verification of a human-interactive system. Despite the use of a seemingly simple target system, a patient controlled analgesia pump, the initial model proved to be difficult for the model checker to verify in a reasonable amount of time. This resulted in a number of model revisions that affected the HFE architectural, representativeness, and understandability goals of the effort. If formal methods are to meet the needs of the HFE community, additional modeling tools and technological developments are necessary.

1 Introduction

Formal methods have proven themselves useful for finding problems in safety-critical systems that could lead to undesirable, dangerous, or disastrous system conditions. The traditional use of formal methods has been to find problems in a system's automation under different operating and/or environmental conditions. However, human operators control a number of safety critical systems and contribute to unforeseen problems. For example, human behavior has contributed to between 44,000 and 98,000 deaths nationwide every year in medical practice [12], 74% of all general aviation accidents [13], and a number of high profile disasters such as the Three Mile Island and Chernobyl accidents [14]. Human factors engineering (HFE) focuses on understanding human behavior and applying this knowledge in the design of systems that depend on human interaction: making systems easier to use while reducing errors and/or allowing recovery from them [17, 19].

Because both HFE and formal methods are concerned with the engineering of robust systems that will not fail under realistic operating conditions, researchers have leveraged the knowledge of both in order to evaluate safety-critical systems. Such synergy has been used for identifying the cognitive precondition for mode confusion and automation surprise [2, 5, 10, 15]; the automatic generation of user interface specifications, emergency procedures, and recovery sequences [6, 9]; and the identification of human behavior sequences (normative or erroneous) that contribute to system failures [3, 8].

When human factors engineers analyze human system interaction, they consider the goals, knowledge, and procedures of the human operator; the automated system and its human interface; and the operational environment. Cognitive work analysis is concerned with identifying constraints in the operational environments that shape the mission goals of the human operator [18]; cognitive task analysis is concerned with describing how human operators normatively and descriptively perform tasks when interacting with an automated system [11, 16]; and modeling frameworks such as [7] seek to find discrepancies between human mental models, human-device interfaces (HDIs), and device automation. In

E. Denney, D. Giannakopoulou, C.S. Păsăreanu (eds.); The First NASA Formal Methods Symposium, pp. 6-15

*Matthew L. Bolton is a PhD student in the University of Virginia's Department of Systems and Information Engineering

†Ellen J. Bass is an Associate Professor in the University of Virginia's Department of Systems and Information Engineering

this context, problems related to human-automation interaction may be influenced by the human operator's mission, the human operator's task behavior, the operational environment, the HDI, the device's automation, and their interrelationships.

In order to allow human factors engineers to exploit their existing modeling tools with the powerful verification capabilities of formal methods to identify potential problems that may be related to human-device interaction, we are developing a computational framework (Figure 1) for the formal modeling of human-interactive systems. This framework utilizes concurrent models of human operator task behavior, human mission directives, device automation, and the operational environment which are composed together to form a larger system model. Inter-model interaction is represented by variables shared between models. Environment variables communicate information about the state of the environment to the device automation, mission, and human task models. Mission variables communicate the mission goals to the human task model. Interface variables convey information about the state of the HDI (displayed information, the state of input widgets, etc.) to the human task model. The human task model indicates when and what actions a human operator would perform on the HDI. The HDI communicates its current state to the device automation via the interface variables. The HDI receives information about the state of the device automation model via the automation state variables.

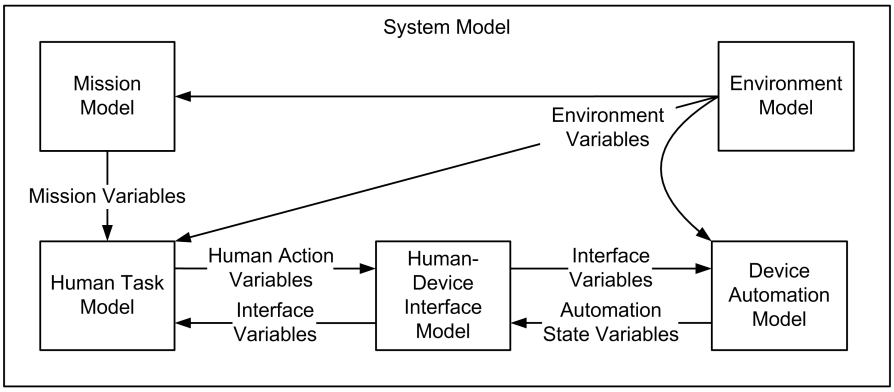

Figure 1: Framework for the formal modeling of a human-interactive system. Arrows between models represent variables that are shared between models. The direction of the arrow indicates whether the represented variables are treated as inputs or output. If the arrow is sourced from a model, the represented variables are outputs of that model. If the arrow terminates at a model, the represented variables are inputs to that model.

To be effective for the human factors community, the analysis framework must support models of the human task and the mission. Because an engineer may wish to rerun verifications using different missions, task models, human-device interfaces, environments, or automation behaviors, these components should remain decoupled (as is the case in Figure 1). Finally, the modeling technique must be capable of representing human-interactive systems with enough fidelity to allow the engineer to perform the desired verification, and do so in reasonable amount of time (this could mean several hours for a small project, or several days for a more complicated one).

In the first instantiation of this framework, we modeled a Baxter Ipump Pain Management System [1], a patient controlled analgesia (PCA) pump that administers pain medication in accordance with health care technician defined constraints. The pump is safety critical as a malfunctioning or miss-programmed pump could potentially overdose a patient. It has a simple HDI (see Figure 2) that can be used to specify a wide range of user mission options (different prescriptions that can be issued). The pump also has automation that checks user inputs and administers treatment.

The end goal of this modeling effort is to perform verifications related to the ability of documentation-derived human task behavior to successfully program the range of available prescriptions into the pump. The initial focus of this effort is to construct and debug the human-device interface and device automation models. An environmental model was not included because of the general stability of the environment in which an actual pump operates. Because this modeling phase was primarily concerned with ensuring that the human-device interface and device automation models were working as intended, the human task model was represented as an unconstrained operator - one that could issue any valid human action at any given time. Finally, the specification used in the verification process was only concerned with ensuring that the models were functioning properly.

Even though the target device being modeled was seemingly simple, the initial model was too difficult for the model checker to process quickly and too complex for it to verify. Thus in order to produce a model that was usable on a reasonable time scale, a number of model revisions ultimately impacted the goals of the framework. This paper discusses these compromises and uses them to draw conclusions about the feasibility of using formal methods to inform HFE.

2 Methods

2.1 The Target System

The Baxter Ipump is an automated machine that allows for the controlled delivery of sedative, analgesic, and anesthetic medication solutions [1]. Solution delivery via intravenous, subcutaneous, and epidural routes is supported. Medication solutions are typically stored in bags locked in a compartment on the back of the pump.

Pump behavior is dictated by internal automation, some of which is dependent on how the pump is programmed by a human operator. Pump programming is accomplished via its HDI (Figure 2) which contains a dynamic LCD display, a security key lock, and eight buttons. When programming the pump, the operator is able to specify all of the following: whether to use periodic or continuous doses of medications (i.e. the mode), whether to use prescription information previously programmed into the pump, the fluid volume contained in the medication bag, the units of measure used for dosage (ml, mg, or μg), whether or not to administer a bolus (an initial dose of medication), dosage amounts, dosage flow rates (for either basal or continuous rates as determined by the mode), the delay time between dosages, and one hour limits on the amount of delivered medication.

During programming, the security key is used to lock and unlock the compartment containing the medication solution. The unlocking and locking process is also used as a security measure to ensure that an authorized person is programming the pump. The start and stop buttons are used to start and stop the delivery of medication at specific times during programming. The on-off button is used to turn the device on and off. The LCD display can be used to choose from a variety of options that affect the way the pump operates. 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. Pressing the up button allows the programmer to scroll through the available options.

When a numerical value is required, the name of the required value 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 or she 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 automation used by the pump is its dynamic checking and restriction of operator entered values. Thus, in addition to having hard limits on

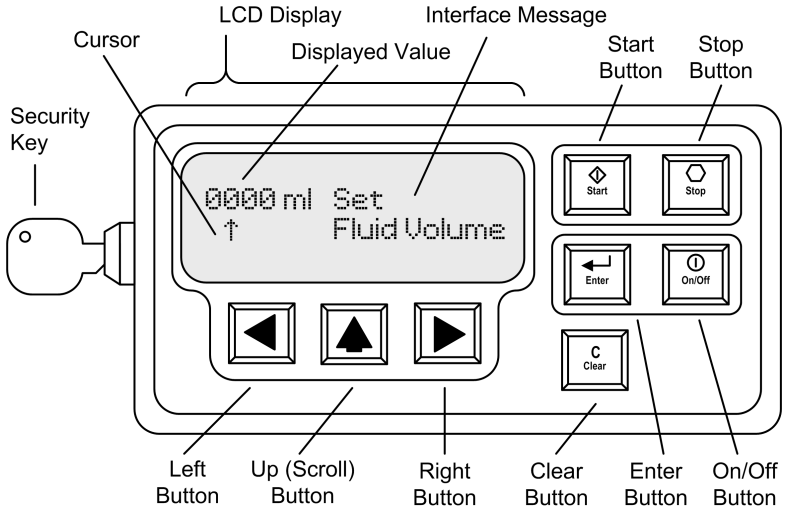

Figure 2: A simplified representation of the Baxter Ipump's human-device interface. Note that the actual pump contains additional controls and information conveyances.

the maximums and minimums a value can assume, the extrema can change dynamically in response to other user specified values.

2.2 Apparatus

All of the models were constructed using the Symbolic Analysis Laboratory (SAL) language [4]. This language was chosen because of its associated analysis and debugging tools, and because it allows for both the asynchronous and synchronous composition of different models (modules using SAL's internal semantics).

All verifications were done using SAL-SMC, the SAL symbolic model checker [1]. Verifications were conducted on a 3.0 gigahertz dual-core Intel Xeon processor with 16 gigabytes of RAM running Redhat Enterprise Linux 5.

2.3 Initial Model

The model that was initially produced was designed to conform to the architecture and design philosophy represented in Figure 2: the mission was represented as a set of viable prescriptions options; the mission, human operator, human-device interface, and device automation were modeled independently of each other; and the behavior of the automated system and HDI models was kept as representative as possible. To limit the scope of the human task model, an unconstrained human operator was constructed that was capable of issuing a valid human action to the human-device interface model at any given time. Further, because the PCA pump generally operates in a controlled environment, away from temperature and humidity conditions that might affect the performance of the pump's automation, no environmental model was included. Finally, because documentation related to the internal workings of the pump was limited, the system automation model was restricted to that associated with the pump programming procedure: behavior that could be gleamed from the operator's manual [1], correspondences with hospital staff, and direct interaction with the pump.

[1] Some model debugging was also conducted using SAL's bounded model checker.

2.4 Verification Specification

Because the model was limited to the prescription programming procedure and human behavior was unrestricted, full verification of the human operator's task behavior was not possible for this instantiation. However, verification could be conducted to ensure that prescriptions could be programmed into the pump. This was done using the SAL-SMC model checker and following specification written in linear temporal logic (state variables are presented in italics):

$$
\begin{array}{llcl}
\mathrm{AG}\neg(& \mathit{InterfaceMessage} & = & \text{TreatmentAdministering} \\
\wedge & \mathit{PrescribedMode} & = & \mathit{EnteredMode} \\
\wedge & \mathit{PrescribedFluidVolume} & = & \mathit{EnteredFluidVolume} \\
\wedge & \mathit{PrescribedPCADose} & = & \mathit{EnteredPCADose} \\
\wedge & \mathit{PrescribedDelay} & = & \mathit{EnteredDelay} \\
\wedge & \mathit{PrescribedBasalRate} & = & \mathit{EnteredBasalRate} \\
\wedge & \mathit{PrescribedOneHourLimit} & = & \mathit{EnteredOneHourLimit} \\
\wedge & \mathit{PrescribedBolus} & = & \mathit{EnteredBolus} \\
\wedge & \mathit{PrescribedContinuousRate} & = & \mathit{EnteredContinuousRate} \quad)
\end{array}
$$

Here, if the model is able to enter a state indicating that treatment is administering (*InterfaceMessage* = emphTreatmentAdministering) with the entered (or programmed) prescription values (variables with the "Entered" prefix) matching the actual prescription values (variables with the "Prescribed" prefix), a counter example is returned.

3 Model Revision

An attempt to verify the initial model resulted in two problems related to the feasibility and usefulness of the verification procedure. Firstly, the SAL-SMC procedure for translating the SAL code into a binary decision diagram (BDD) took excessively long (more than 24 hours), a time frame impractical for model debugging. Secondly, the verification process which followed the construction of the BDD, eventually ran out of memory, thus not returning a verification result. Thus, a number of revisions were required to make the model more tractable, some of which compromised the architectural, representation, and interpretability goals of the effort.

3.1 Model Coordination

Even before the initial verification, some additional model infrastructure was required in order to ensure that human operator actions were properly recognized by the HDI model. In an ideal modeling environment, human action behavior originating from the human operator model would be able to have both an asynchronous and synchronous relationship with the HDI model. Synchronous behavior would allow the HDI model to react to user actions in the same transition in which they were issued/performed by the human operator model. However, both the human operator and HDI models operate independently of each other, and may have state transitions that are dependent on internal or external conditions that are not directly related to the state of the other model. This suggests an asynchronous relationship. SAL only allows models to be composed with each other asynchronously or synchronously (but not both). Thus, it was necessary to adapt the models to support features associated with the unused composition.

Because of independent nature of the models in the framework, asynchronous composition was used to compose the human operator and HDI models. This necessitated some additional infrastructure to prevent the human operator model from issuing user inputs before the HDI model was ready to interpret them and to prevent the human operator model from terminating a given input before the interface could

respond to it. This was accomplished through the addition of two Boolean variables: one indicating that input had been submitted (henceforth called Submitted) and a variable indicating the interface was ready to receive actions (henceforth called Ready). This coordination occurred as follows:

- If Ready is true, the human operator module sets one or more of the human action variables to a new input value and sets Submitted to true.
- If Ready and Submitted are true, the human-device interface module responds to the values of the human action variables and sets Ready to false.
- If Ready is not true and Submitted is true, the human operator module sets Submitted to false.
- If Ready and Submitted are both not true and the automated system is ready for additional human operator input, the human-system interface module sets Ready to true.

3.2 Representation of Numerical Values

In order to reduce the time needed to convert the SAL-code model to a BDD, a number of modifications were made to represent model constructs in a way that could be more readily processed by the model checker. As such, the modifications discussed here did not ultimately make the BDD representation of the model smaller, but merely expedite its construction.

3.2.1 Redundant Representation of Values

The dichotomy between the HDI and device automation models resulted in a situation where two different representations of user specified numerical values were convenient. Because the HDI required the human operator to enter values by scrolling through the available values for individual digits, it was conceptually simple to model these values as an array of integer digits in the HDI model. However, because the system automation was concerned with dynamically checking limits and using entered values to compute other values, a numerical representation of the actual value was more convenient for the automated system model.

Because this redundancy of information necessitated additional effort on the part of the BDD translator, it was remedied by eliminating the digit array representations. To accommodate this, a variety of functions needed to be created in order to allow actions from the human task model to dynamically change individual digits within a value.

3.2.2 Real Numbers and Integers

In the original model, all numerical values were represented as real values with restricted ranges. This was done because most user specified values were either integers or floating point numbers (precise to a single decimal point). Representing values this way proved especially challenging for the BDD translator. Thus, all values were modified so that they could be represented as restricted range integers. For integer variables representing floating point numbers, this meant that the model value was ten times the value it represented.

3.2.3 Variable Ranges

In the initial model, the upper bound on the range of all value based variables was set to the maximum amount that could be programmed into the pump: 99999[2]. However, in order to reduce the amount

[2]All lower bounds were set to 0.

of work required for the BDD conversion, the range of each numerically valued variable was given a specific upper bound that corresponded to the actual maximum value that a human operator could assign to it.

3.2.4 Model Reduction

In order to reduce the size of the model, a variety of elements were removed. In all cases these reductions were meant to reduce the number of state variables in the HDI or Device Automation models (slicing), or reduce the range of values a variable could assume (data abstraction). Unfortunately, each of these reductions also affected what user tasks could ultimately be modeled and thus verified in future work. All of the following reductions were undertaken:

- In the original model, the mission model could generate a prescription from the entire available range of valid prescriptions. The scope of this was significantly reduced so that only several prescription options were generated . While this significantly reduced the number of model states, it significantly reduced the number of prescriptions that could be used in verification procedures.
- In the original model, the HDI model would allow the operator to select what units to use when entering prescriptions (ml, mg, or μg). This was removed from the model, with the most general unit option (ml) becoming standard. This reduced the number of interface messages in the model, allowed for the removal of several variables (those related to the unit option selection, and solution concentration specification), and reduced the ranges required for several numerical values related to the prescription. This also eliminated the option of including unit selection task behavior in future work.
- In the original model, both the HDI and device automation models encompassed behavior related to the delivery of medication solution during the priming and bolus administration procedures. During priming, the HDI allows the operator to repeatedly instruct the pump to prime until all air has been pushed out of the connected tubing. During bolus administration, the HDI allows the operator to terminate bolus infusion by pressing the stop button twice. This functionality was removed from the models, thus eliminating interface message states and numerical values indicating how much fluid had been delivered in both procedures. This removed the option of incorporating task behavior related to pump priming and bolus administration.
- The original model mimicked security features found in the original device which required the operator to unlock and lock the device on startup and enter a security code. This functionality was removed from the model which reduced the number of interface messages in the model and removed the numerical variable associated with entering the security code (a 0 to 999 ranged variable). This eliminated the possibility of modeling human task behavior related to unlocking and locking the pump as well as entering the security code.
- In the original model, the interface message could automatically transition to being blank: mimicking the actual pump's ability to blank its screen after three seconds of operator inactivity. Because further operator inaction would result in the original device issuing a "left in programming mode" alert, a blank interface message could automatically transition to an alert issuance. This functionality was removed from the model, eliminating several interface messages as well as variables that kept track of the previous interface message. Thus, the ability to model operator task response to screen blanking and alerts was also removed.

While these reductions resulted in a model that was much smaller and more manageable than the original, and one that could still be used concurrently with operator task behavior models for programming pump

prescriptions, the ability to model some of task behaviors originally associated with the device had to be sacrificed.

3.2.5 Results

As a result of the revisions discussed in this section, the model was able to complete the entire verification procedure in approximately 5.9 hours, with the majority of that time (5.4 hours) being required to create the BDD representation. Because of the nature of the specification, a counterexample was produced which was used to verify that the model was behaving as intended[3].

4 Discussion

The modeling framework discussed in this paper had three primary goals:

1. Model constructs needed to be represented in a way that was intuitive to a human factors engineer who would be building and evaluating many of the models;
2. The sub-models would remain decoupled and modular (as seen in Figure 1) in order to allow for interchangeability of alternative sub-models; and
3. The models constructed around the framework needed to be capable of being verified in a reasonable amount of time.

While this effort was able to produce verifiable model of a human-interactive, safety-critical system, it did so at the cost of compromising all three of these goals. We discuss how each of these goals was violated and how related issues might be addressed.

4.1 Goals 1: Model Intuitiveness

Many of the model revisions were associated with representing model constructs in ways that were more readily interpretable by the model checker rather than the HFE modeler. These primarily took the form of converting floating point and digit array values as integers. There are two potential ways to address this issue. One solution would be to improve the model checkers themselves. Given that the modifications would not actually change the number of reachable states in the system, this would suggest that the model checker need only optimize the BDD conversion algorithms.

Alternatively, additional modeling tools could be used to help mitigate the situation. Such tools could allow human factors engineers to construct or import HDI prototypes, and translate them into model checker code. This would allow the unintuitive representations necessary for ensuring a model's efficient processing by the model checker to be removed from the modeler's view.

Tools could also be created to translate counterexample data into visualizations of HDI use cases. This would prevent the modeler from having to understand the code produced by the translation when examining counterexample data.

4.2 Goal 2: Decoupled Sub-models

Because the communication protocol used to coordinate human actions between the HDI and human task models assumes a particular relationship between variables shared by these models, they are more

[3]The completed model, SAL output, and counterexample can be found at `http://cog.sys.virginia.edu/formalmethods/`

tightly coupled than they would be otherwise. Unless SAL can be made to support both asynchronous and synchronous relationships between models more elegantly, there is little that can be done to eliminate the coordination infrastructure.

However, a solution may be found in an additional level of abstraction. Should a toolset exist for translating an HDI prototype into model checking code, then this translation could handle the construction of the coordination protocol, making this process effectively invisible to the modeler.

4.3 Goal 3: Model Verifiability

One of the reasons this particular modeling effort was so challenging was because the target system was dependent on a large number of user specified numerical values, all of which had very large acceptable ranges. This resulted in the scope of the model being reduced to the point where it could no longer be used for verifying all of the original human operator task behaviors.

Given the simplicity of the device that was modeled, it is clear that systems with a larger number of operator specified numerical values would be more challenging to verify, if not impossible. While the use of bounded model checkers may provide some verification capabilities for certain systems, there is little that can be done without advances in model checking technology and computation power.

However, it is possible that, had the target system been more concerned with procedural behaviors and less on the interrelationships between numerical values, the system model would have been much more tractable. Future work should identify the properties of human-interactive systems that lend themselves to be modeled and verified using the framework discussed here.

5 Conclusion

The work presented here has shown that it is possible to construct models of human-interactive systems for use in formal verification processes. However, this success was the result of a number of compromises that produced a model that was not as representative, understandable, or modular as desired. Thus, in order for formal methods to become more useful for the HFE community, the verification technology will need to be able to support a more diverse set of systems. Further, new modeling tools may be required to support representations that human factors engineers use. These advances coupled with efficient means of representing realistic human behavior in the models will ultimately allow formal methods to become a more useful tool for human factors engineers working with safety critical systems.

Acknowledgement

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.

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 us understand the functionality of the Baxter Ipump and providing them with the necessary documentation, training materials, and device access.

References

[1] Baxter Heath Care Corporation. *Ipump Pain Management System Operator's Manual*, 1995.

[2] J. Crow, D. Javaux, and J. Rushby. Models and mechanized methods that integrate human factors into automation design. In *Proceedings of the International Conference on Human-Computer Interaction in Aeronautics*, 2000.

[3] P. Curzon, R. Ruksenas, and A. Blandford. An approach to formal verification of human-computer interaction. *Formal Aspects of Computing.*, 19(4):513–550, 2007.

[4] L. De Moura, S. Owre, and N. Shankar. The SAL language manual. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 2003.

[5] A. Degani. *Modeling Human-machine Systems: On Modes, Error, and Patterns of Interaction*. PhD thesis, Georgia Institute of Technology, 1996.

[6] A. Degani, M. Heymann, and I. Barshi. A formal methodology, tools, and algorithm for the analysis, verification, and design of emergency procedures and recovery sequences. *NASA Internal White Paper*, 2005.

[7] A. Degani and A. Kirlik. Modes in human-automation interaction: initial observations about a modeling approach. In *IEEE International Conference on Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century.*, volume 4, 1995.

[8] R.E. Fields. *Analysis of Erroneous Actions in the Design of Critical Systems*. PhD thesis, University of York, 2001.

[9] M. Heymann and A. Degani. Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm. *Human Factors*, 49(2):311–330, 2007.

[10] D. Javaux and P.G. Polson. A method for predicting errors when interacting with finite state machines. *Reliability Engineering and System Safety*.

[11] B. Kirwan and L.K. Ainsworth. *A Guide to Task Analysis*. Taylor and Francis, 1992.

[12] L.T. Kohn, J. Corrigan, and M.S. Donaldson. *To Err is Human: Building a Safer Health System*. National Academy Press, 2000.

[13] N.C. Krey. 2007 Nall report: accident trends and factors for 2006. Technical report, 2007. `http://download.aopa.org/epilot/2007/07nall.pdf`.

[14] C. Perrow. *Normal Accidents*. Basic Books New York, 1984.

[15] J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, 2002.

[16] J.M. Schraagen, S.F. Chipman, and V.L. Shalin. *Cognitive Task Analysis*. Lawrence Erlbaum Associates, 2000.

[17] N. Stanton. *Human Factors Methods: A Practical Guide for Engineering and Design*. Ashgate Publishing, Ltd., 2005.

[18] K.J. Vicente. *Cognitive Work Analysis: Toward Safe, Oroductive, and Healthy Computer-based Work*. Lawrence Erlbaum Assoc Inc, 1999.

[19] C.D. Wickens, J. Lee, Y.D. Liu, and S. Gordon-Becker. *Introduction to Human Factors Engineering*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2003.