Generating Erroneous Human Behavior from Strategic Knowledge in Task Models and Evaluating its Impact on System Safety with Model Checking

Matthew L. Bolton, *Member*, *IEEE*, and Ellen J. Bass, *Senior Member*, *IEEE*

***Abstract*—Human-automation interaction, including erroneous human behavior, is a factor in the failure of complex, safety-critical systems. This paper presents a method for automatically generating formal task analytic models encompassing both erroneous and normative human behavior from normative task models, where the misapplication of strategic knowledge is used to generate erroneous behavior. Resulting models can be automatically incorporated into larger formal system models so that safety properties can be formally verified with a model checker. This allows analysts to prove that a human-automation interactive system (as represented by the formal model) will or will not satisfy safety properties with both normative and generated erroneous human behavior. Benchmarks are reported that illustrate how this method scales. The method is then illustrated with a case study: the programming of a patient-controlled analgesia pump. In this example, a problem resulting from a generated erroneous human behavior is discovered. The method is further employed to evaluate the effectiveness of different solutions to the discovered problem. The results and future research directions are discussed.**

***Index Terms*—Task analysis, formal methods, model checking, human error, human-automation interaction, system safety.**

I. Introduction

COMPLEX, safety-critical systems involve the interaction of automated devices and goal-oriented human operators in a dynamic environment. Human-automation interaction (HAI) [1], [2] is particularly important to the operation of safety-critical systems. Erroneous human behavior [3], where the human operator does not follow the normative procedures for interacting with a system, is often associated with failures. HAI research has produced a number of analysis techniques and design tools that can be used to address this problem. In particular, emerging model-driven design and analysis techniques [4], [5] use models of the automation, human-device interfaces (HDIs), human task behavior, human cognition, and/or environmental conditions to provide guarantees about the performance of the system using formal methods.

Formal methods are a set of languages and techniques for the modeling, specification, and verification of systems (usually computer hardware or software) [6]. There are many different ways that a model can be formally verified. Model checkers are computer software tools that support automatic verification analyses. In model checking, an automated process verifies whether or not a formal model of a system satisfies a set of desired properties (a specification) [7]. A formal model describes a system as a set of variables and transitions between variable values (states). Specification properties assert properties that the analyst wants to be true with the system, usually using a temporal logic. Verification is the process of proving that the system meets the properties in the specification. Model checking does this by exhaustively searching a system's statespace to determine if the specification holds. If there is a state in the model that violates the specification, a single counterexample (execution trace) is produced which represents a counterproof: it identifies the incremental model states that led up to the violation.

Formal, model-based approaches have been used to evaluate HAI in a number of different ways (see [5] for a survey). These approaches include different methods for evaluating the potential impact of erroneous human behavior on system safety. Poor usability conditions in HDIs can be investigated using formal verification [8]–[11]. Formal verification can be used to find potential mode confusion[1] [14]–[23], which can produce erroneous human behaviors [13]. Formal system models can include a model of cognitively-driven human behavior [24]–[27], and formal verification can be used to evaluate when the modeled cognitive factors could result in erroneous human behavior causing a problem [28]–[34]. Finally, task analytic behavior models (products of a cognitive task analysis [35], [36]) can be included in the formal system model and formal verification can be used to evaluate the impact of the modeled behavior (which can be normative or erroneous) on system safety [37]–[50]. While there are distinct tradeoffs between all of these approaches (see section VI-B), this last one is advantageous in that it explicitly models the impact of human behavior, gives analysts a clear indication of what the human operator was attempting to do when problems occurred, and uses models that are commonly employed by human factors and systems engineers [51]. However, it may not scale as well as the other methods in some circumstances [43], [51].

This paper presents a method which extends the formal verification work on task analytic behavior models. This new approach can be used to automatically generate erroneous human behavior based on the physical manifestation of a human operator's misapplication of strategic knowledge (the

Matthew L. Bolton is with the Department of Mechanical and Industrial Engineering at the University of Illinois at Chicago, Chicago, IL 60607 USA e-mail: mbolton@uic.edu

Ellen J. Bass is with the College of Information Science and Technology and the College of Nursing and Health Professions, Drexel University, Philadelphia

Manuscript received XXXXX; revised XXXXX

[1]Mode confusion is a HAI issue which occurs when the human operator is unable to keep track of the state or mode of the device automation [12], [13].

knowledge the human operator uses to determine what the appropriate plan is to achieve goals) that can occur as a result of attention failures. The impact of the resulting behavior on system safety can be evaluated with model checking. To motivate this work, we first discuss erroneous human behavior taxonomies and report how they have been used with task behavior models and formal verification to evaluate safety-critical systems. We then describe our new method. We present analysis results which show how it scales. We illustrate the method with a case study: the programming of a patient controlled analgesia pump. Finally we discuss our method and opportunities for future work.

A. Taxonomies of Erroneous Human Behavior

There are a number of different ways to classify and model erroneous human behavior [52], [53]. Of the most relevance to this work are Hollnagel's phenotypes of erroneous action [3] and Reason's Generic Error Modeling System [54].

Hollnagel [3] classified erroneous human behaviors based on their phenotype: how an erroneous behavior observably deviates from a normative plan (task) of actions. All phenotypes of erroneous human behavior are constructed from zero-order phenotypes, those that represent deviations of behavior for a single human action in a plan: prematurely starting an action, delaying the start of an action, prematurely finishing an action, delaying the completion of an action, omitting an action, jumping forward (performing an action that should be performed later), jumping backward (performing a previously performed action), repeating an action, and performing an unplanned action (an intrusion).

Alternatively, Reason [54] classified erroneous behaviors based on their cognitive causes, their genotypes. Reason identified a class of erroneous behaviors called slips. Slips occur when a person fails to notice a system or environmental condition (due to a failure of attention) and thus does not perform a task normatively. A person may be interrupted or may otherwise not be attending to the proper information and omit actions. A person may erroneously repeat actions after losing his place in a task. A person may also have his attention "captured" by something else (external or internal) which results in him performing (committing) inappropriate actions either in addition to or instead of appropriate ones. From a completely observable perspective, this means that a slip can manifest as: (a) An omission – the failure to perform all or part of an activity; (b) A repetition – the repeated performance of an activity or action; or (c) A commission – the inappropriate performance of an activity or action.

B. Erroneous Behavior, Task Models, and Formal Methods

When designing for HAI, task analytic methods can be used to describe normative human behavior [35]. The resulting models represent the mental and physical activities human operators use to achieve goals with the system. These models are often hierarchical: activities decompose into other activities and, at the lowest level, atomic actions. Strategic knowledge controls when activities can execute and modifiers between activities or actions control how they execute in relation to each other. Many task analytic models can be represented with discrete graph structures [45], [55]–[57].

Researchers have investigated how erroneous human behavior can be manually, systematically incorporated into normative task analytic behavior models for use in analyses. The majority of this work has focused on identifying ways of inserting Hollnagel's phenotypes of erroneous action [3] into normative task behavior models [47], [58]–[62]. Paternò and Santoro [59] presented a different approach for modeling higher order erroneous behaviors more akin to the physical manifestation of Reason's [54] slips. In this technique erroneous behaviors could occur due to high level activities executing at the wrong time or failing to execute at the correct time.

Because they can be represented discretely, task analytic models can be used to include human behavior in formal system models along with other system elements including device automation, HDIs, and the operational environment [37]–[41], [43]–[50]. This allows system safety properties to be verified in light of the modeled human behavior which could include any erroneous behaviors incorporated into the model using the above techniques.

Bolton et al. [51] developed a more automated approach. A task structure capable of generating erroneous human behaviors based on Hollnagel's zero-order phenotypes can be automatically incorporated into normative human task behavior models by replacing each action in the original hierarchy. The number of generated erroneous behaviors (zero-order phenotypes) is limited by an analyst-specified upper bound. The resulting task behavior models are automatically translated into formal system models [45], [51]. A model checker is used to evaluate the impact of both the original (normative) and generated (erroneous) behavior on system safety properties. This method has a distinct advantage over the other erroneous behavior generation techniques in that it allows erroneous behaviors that may not have been anticipated by analysts to be considered. While this technique has proven itself useful for finding potential problems in human-automation interactive systems, a large upper bound on the number of erroneous acts would be required to generate the activity level erroneous behaviors explored by Paternò and Santoro [59]. Such large upper bounds will generate erroneous behavior patterns that analysts may not find interesting. Further, as the upper bound on the number or erroneous acts increases, both the number of states and the safety property verification time increase exponentially. This is problematic because, for a realistic system, the size of the model may exceed the resources available to the model checker on the analysis machine or verification may take too long to complete.

C. Objectives

For a system that can be modeled formally and whose human task behavior can be represented using a hierarchical task analytic modeling formalism, we have developed a method that allows analysts to automatically evaluate the impact of erroneous behaviors like those discussed by Paternò and Santoro [59] on system safety. To accomplish this, we modify the way strategic knowledge is interpreted in task

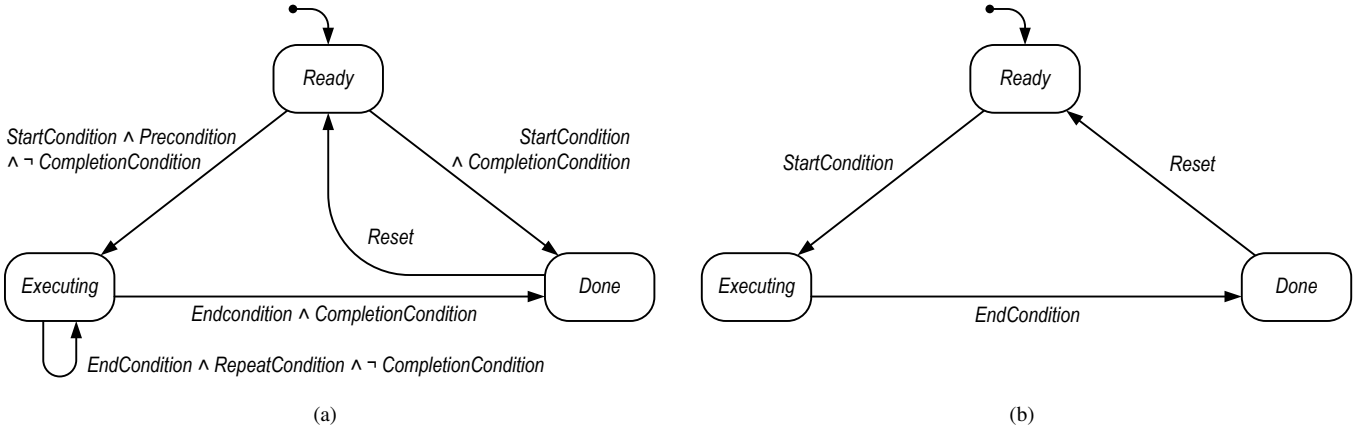


Fig. 1. Formal semantics of an EOFM (a) activity's and (b) action's execution state presented as finite state transition systems [45]. States are represented as rounded rectangles. Transitions appear as arrows between states that are labeled with Boolean expressions. Arrows starting with a dot point to initial states.

analytic behavior models to replicate Reason's [54] slips and evaluate their impact on the system using model checking. We first discuss the infrastructure in which this technique was implemented. We then describe our method. Benchmark results are presented which show how the method scales. A patient-controlled analgesia (PCA) pump application is then used to illustrate how our method can be used to find problems in a human-automation interactive system. Results and avenues of future work are discussed.

II. EOFM and the Formal Verification of HAI

We introduced an architectural framework [43], [44] to evaluate HAI formally using task analytic models of human behavior, models of human missions (i.e. goals), HDIs, device automation, and the operational environment. Human task models are created using an intermediary language called Enhanced Operator Function Model (EOFM) [45], an XML-based human task modeling language derived from the Operator Function Model (OFM) [56], [63]. EOFMs are hierarchical and heterarchical representations of goal driven activities that decompose into lower level activities, and finally, atomic actions. Actions are typically observable, singular ways the human operator can interact with the device. However, EOFM also supports local variables, where the assignment of a value to a local variable can represent a cognitive and perceptual action. A decomposition operator specifies the temporal relationships between and the cardinality of the decomposed activities or actions (when they can execute relative to each other and how many can execute).

EOFMs express strategic knowledge explicitly as conditions on activities. Conditions can specify what must be true before an activity can execute (preconditions), if it can repeat execution (repeat conditions), and what is true when it completes execution (completion conditions).

EOFMs can be represented visually as a tree-like graph [64] (see examples in Figs. 3 and 8). Actions are rectangles and activities are rounded rectangles. An activity's decomposition is presented as an arrow, labeled with the decomposition operator, that points to a large rounded rectangle containing the decomposed activities or actions. In the work presented here, four of the nine decomposition operators [45] are used: (a) *ord* – all activities or actions in the decomposition must execute in the order they appear; (b) *or_seq* – one or more of the activities or actions in the decomposition must execute and only one can execute at a time; (c) *optor_par* – zero or more of the activities or actions in the decomposition must execute and their execution can overlap; and (d) *xor* – exactly one activity or action in the decomposition must execute.

Conditions (strategic knowledge) on activities are represented as shapes or arrows (annotated with the logic) connected to the activity that they constrain. The form, position, and color of the shape are determined by the type of condition. A precondition is a yellow, downward-pointing triangle; a completion condition is a magenta, upward-pointing triangle; and a repeat condition is an arrow recursively pointing to the top of the activity. More details can be found in [45].

EOFM has formal semantics which specify how an instantiated EOFM model executes (Fig. 1). Specifically, each activity or action has one of three execution states: waiting to execute (*Ready*), executing (*Executing*), and done (*Done*). An activity or action transitions between each of these states based on its current state; its start condition (*StartCondition* – when it can start executing based on the state of its immediate parent, its parent's decomposition operator, and the execution state of its siblings); its end condition (*EndCondition* – when it can stop executing based on the state of its immediate children in the hierarchy and its decomposition operators); its reset condition (*Reset* – when it can revert to *Ready* based on the execution state of its parents); and, for an activity, the activity's strategic knowledge (the *Precondition*, *RepeatCondition*, and *CompletionCondition*). See [45] for more details.

Instantiated EOFM task models can be automatically translated [45] into the language of the Symbolic Analysis Laboratory (SAL) [65] using the EOFM formal semantics. The translated EOFM can then be integrated into a larger system model using a defined architecture and coordination protocol [43], [45]. Formal verifications are performed on this complete system model using SAL's Symbolic Model Checker (SAL-SMC). Any produced counterexamples can be visualized and evaluated using EOFM's visual notation (see [64]).

We next discuss the design philosophy behind our erroneous behavior generation method and describe how it can be automatically incorporated into this infrastructure.

III. Erroneous Human Behavior Generation

In Reason's taxonomy [54], slips occur due to attention failures and can manifest as omissions, repetitions, and commissions. Our erroneous human behavior generation method models these slips as occurring due to a human operator not properly attending to the strategic knowledge contained in pre, repeat, and completion conditions. This was done by making changes to the EOFM's formal semantics. In this design, the original semantics (Fig. 1) were given additional transitions (Fig. 2) to describe when an activity could erroneously switch between execution states. Each new transition represents the erroneousness analog of a non-erroneous transition, where the erroneous transition is conditioned on the same start or end condition as well as the negation of any strategic knowledge (pre, completion, or repeat condition) used by the non-erroneous transition. This allows for the generation of slips as omissions, repetitions, and commissions. Further, the transitions were designed to limit the number of erroneous behaviors considered in any given analysis.

A. Omissions

Omissions occur when the human operator fails to attend to when he or she should perform an activity and thus does not perform that activity [54]. Thus, an omission can occur in two ways: (1) The human is not attending to when the activity should be performed and thus does not perform it; or (2) The human is not attending to the performance of an activity and finishes it too early.

To replicate the first condition, our method adds an erroneous *Executing* to *Done* transition to the EOFM formal semantics (Fig. 2). This encapsulates the circumstances where the human operator is not paying attention to when the activity should be performed (not properly attending to the environmental and system conditions encoded into the *CompletionCondition*) and does not perform it.

To represent the second condition, our methods includes an erroneous *Ready* to *Done* transition (Fig. 2). This models the situations where the human operator is not paying attention to when an already executing activity should continue executing (fails to correctly attend to the *CompletionCondition*) and stops executing the activity prematurely.

These transition are only relevant if the activity has a *CompletionCondition*.

B. Repetitions

An erroneous repetition occurs when a human operator looses his or her place when executing an activity and erroneously repeats it [54]. Our erroneous behavior generation methods models this as an erroneous *Executing* to *Executing* transition (Fig. 2). This models a situation where a human operator is not properly attending to the systems and environmental conditions encoded in the *RepeatCondition* and the

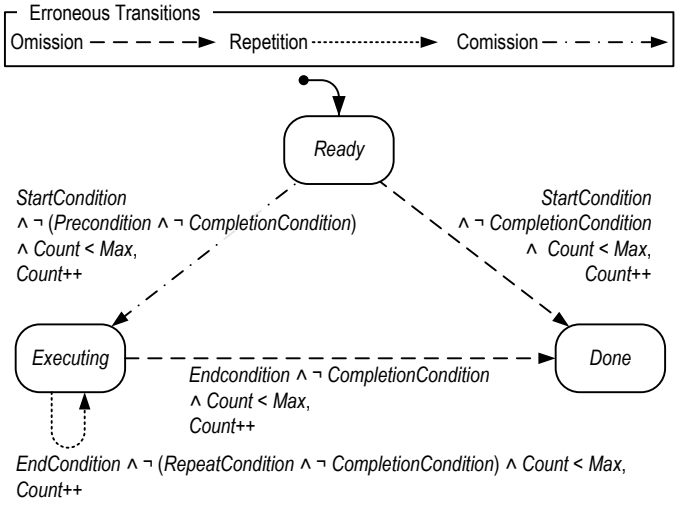


Fig. 2. Additional EOFM activity formal semantic transitions for generating erroneous behavior beyond the normative transitions shown in Fig. 1(a).

CompletionCondition and erroneously repeats the execution of the activity. This transition is only relevant if the activity has a *RepeatCondition*.

C. Commissions

A commission occurs when a human operator has his or her attention captured by something else in the environment and erroneously executes an activity [54]. Our method generates this via an erroneous *Ready* to *Executing* transition. This transition is conditioned on situations where a human operator's attention is captured by states other than those associated with the correct evaluation of the *Precondition* and *CompletionCondition*. This transition is only relevant if the activity has a *Precondition* or *CompletionCondition*.

D. Controlling the Number of Erroneous Transitions

Too many erroneous transitions could result in an unbounded human task behavior model which would defeat the benefit of having a task model. Thus, the analyst can limit the number of erroneous transitions using an upper bound (*Max*). A variable (*Count*) keeps track of the number of erroneous transitions. An erroneous transition can only be undertaken if the current number of erroneous transitions is less than the maximum ($Count < Max$). Every time an erroneous transition occurs, *Count* is incremented by one (*Count*++).

E. Implementation

Our Java-based EOFM to SAL translator [45] was modified to optionally incorporate these erroneous transitions into the translated SAL version of an instantiated EOFM. The translator takes the maximum number of erroneous acts (*Max*) as input from the analyst. *Max* is represented as a constant and an enumerated type is used to represent the range of the possible number of erroneous transitions. The human operator's formal representation has a local variable for the number of erroneous transitions that have occurred (*Count*).

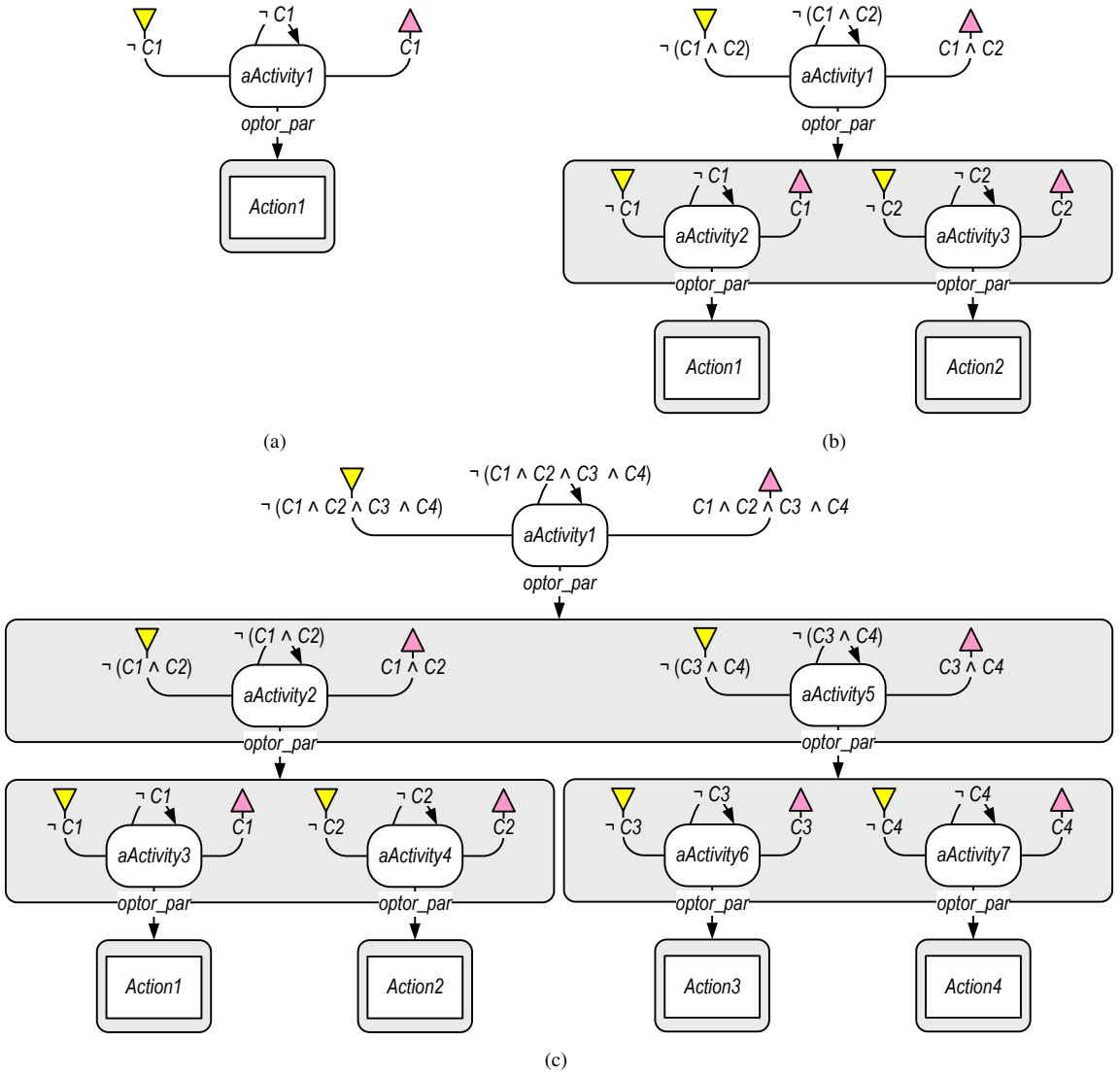


Fig. 3. Instantiated EOFM normative task structures used as inputs to verification benchmark experiments. Activities begin with the letter "a" and atomic actions do not. (a) The human operator must perform *Action1* until condition *C1* is true. (b) The human operator must perform *Action1* and *Action2* until *C1* and *C2* are true respectively. (c) The human operator must perform *Action1*, *Action2*, *Action3*, and *Action4* until *C1*, *C2*, *C3* and *C4* are true respectively.

When writing the transition logic for each activity, this implementation adds transitions (guards and variable assignments) for each of the dotted lines in Fig. 2[2]. The variable assignment for each erroneous transition is identical to its non-erroneous counterpart in the SAL code except that it adds an assignment that increments the erroneous transition count.

When the translated model is incorporated into a larger formal system model and model checking is performed, the returned counterexample may not be of interest to the analyst for a variety of reasons (such as the infeasibility for an intervention). In this situation, the analyst may wish to rerun the analysis without considering a specific erroneous behavior. The analyst can accomplish this by manually editing the translated erroneous human behavior model to remove the undesired erroneous transition. Given the way EOFM formal semantics are implemented in SAL, each erroneous transition is represented by a single guard (the condition on the transition) and a set of assignments under the guard. Thus, to remove any given erroneous transition, an analyst need only delete or comment out the associated guard and assignments.

[2]This is in addition to the other, normative (un-dotted) transitions from Fig. 1 that are already produced by the translator [45].

IV. Benchmarks

One of the main concerns with model checking analyses is statespace explosion [7]. Thus, benchmarks were collected to determine how the erroneous behavior generation process affected the scalability of the method in terms of statespace size and verification time. The erroneous behavior generation can add complexity to an instantiated EOFM task behavior by adding additional transitions to the previously normative model, thus allowing for more reachable states. The number of erroneous transitions is influenced by two factors: the maximum number of erroneous transitions (*Max*) and the number of strategic knowledge conditions (pre, repeat, and completion conditions) in the EOFM task analytic model. Thus, both factors are accounted for in the benchmark experiments.

To account for the complexity associated with the number of strategic knowledge conditions, we constructed three different instances of EOFM normative task behavior models (Fig. 3), where the number of strategic knowledge conditions increases from the simplest to the most complex: Fig. 3(a) has 3, Fig. 3(b) has 9, and Fig. 3(c) has 21. All three of these models assume that the human operator is interacting with a simple system in which he is trying to make conditions become true. The human must perform *Action1* until the condition *C1* is true (in all three tasks in Fig. 3), *Action2* until the *C2* is true (for the tasks in Figs. 3(b) and (c)), *Action3* until the condition *C3* is true (for the task in Fig. 3(c)), and *Action4* until the condition *C4* is true (for the task in Fig. 3(c)). An *optor_par* decomposition operator (zero or more of the activities/actions in the decomposition must execute and their execution can overlap) was used in all of the decompositions in all three task models because it is the decomposition operator associated with largest number of model states [45].

To account for the complexity associated with the maximum number of allowable erroneous transitions (*Max*), we varied the maximum number of erroneous acts (between 0 and 16) for each of the task structures in Fig. 3. The SAL translator was used to create a formal model for each unique combination, where each translated model was paired with models that updated the state of the appropriate conditions (*C1*, *C2*, *C3*, and *C4*) in response to human actions (see Fig. 4).

SAL-SMC was used to formally verify each model against a valid specification ((1) for models using the task in Fig. 3(a) and (2) for models using the tasks in Figs. 3(b) and (c)) on a computer workstation with 16 gigabytes of RAM, a 3.0 gigahertz dual-core Intel Xeon processor, and the Ubuntu 9.04 desktop. Both the total number of visited states and the verification times that were reported by SAL-SMC are presented in Fig. 5.

$$\mathbf{G}\neg(aActivity1 = Executing \wedge Action1 = Executing) \quad (1)$$

$$\mathbf{G}\neg(aActivity1 = Executing \wedge aActivity2 = Executing) \quad (2)$$

Both the number of visited states and the verification time increased linearly with *Max* (see Fig. 5). The correlation between *Max* and the number of visited states yielded $R^2 \approx 1$ for all three data sets. The correlation between *Max* and the verification times for the task models in Figs. 3(a), 3(b), and 3(c) produced R^2 values of 0.96, 0.93, and 0.82 respectively.

The data and correlation measures (Fig. 5) indicate that verification time does not scale as perfectly linearly with *Max* as the statespace does. Further, there are some interesting inconsistencies between data sets. For example, in Fig. 5(b), verification times are very close between sequential even and odd values of *Max*; in Fig. 5(c), verification time varies significantly between even and odd numbers of *Max*, with the verification taking longer for odd numbers of *Max*; and in Fig. 3(a), there is no such pattern between even and odd numbers of *Max*. There are a number of reasons for why these variations could occur. Firstly, there are many operating system processes that are concurrently executing with the model checker which could add variation to the verification time results. Secondly, the nature of the model checker itself will add variation to the verification time. As part of its verification process, SAL creates a symbolic representation of the input model optimized for checking the specification property. Further, SAL may employ different search algorithms to produce results as quickly as possible. As a result of this, verification times can vary between even similar models. A deeper exploration of this anomaly would require investigating the algorithms SAL uses in its model checking, which exceeds the scope of this paper. However, even with these sources of variance, verification time did tend to increase linearly with *Max*. In the context of model checking, this is a positive result given that the erroneous behavior generation method is not exhibiting combinatorial explosion, the primary limiting factor for model checking analyses [7].

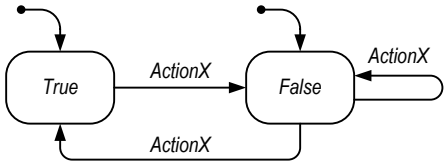


Fig. 4. Finite state transition description for *CX* where *X* can be 1, 2, 3 or 4. *CX* can start out being either true or false. If *CX* is true and the human operator performs *ActionX*, *CX* becomes false. If *CX* is false and the human operator performs *ActionX*, then *CX* can either become true or remain false.

V. Application

To illustrate how this method can be used to discover potential system problems, we present a PCA pump programming application, extended from [43], [44], [66]. A PCA pump is a medical device that allows patients to control the delivery of pain medication based on a prescription programmed into it by a human practitioner. The HDI for the device (Fig. 6) contains a dynamic LCD and eight buttons. This pump accepts three prescription values: a PCA dosage in ml, a minimum delay between dosages in minutes, and a one hour dosage limit in ml. The device gives practitioners the option to review prescriptions before administering treatment.

The practitioner uses the HDI to program prescription parameters. The "Start" and "Stop" buttons start and stop the delivery of medication (stop must be pressed twice) at certain times during programming. The "On-Off" button is used to turn the device on (when pressed once) and off (when pressed twice). The LCD displays information and allows the practitioners to specify prescription values. A prescription value's name is displayed on the LCD and the value is presented with the cursor under one of its digits. The practitioner can change the position of the cursor by pressing the left and right buttons. The practitioner can press the up button to scroll through the different digit values available at the current cursor position. The "Clear" button sets the displayed value to zero. The enter button is used to confirm values and treatment options.

A. Formal Modeling

All of the formal models were constructed using the Symbolic Analysis Laboratory (SAL) language [65][3]. The formal

[3]All of the models presented in this paper are available at http://sys.uic.edu/resources/.

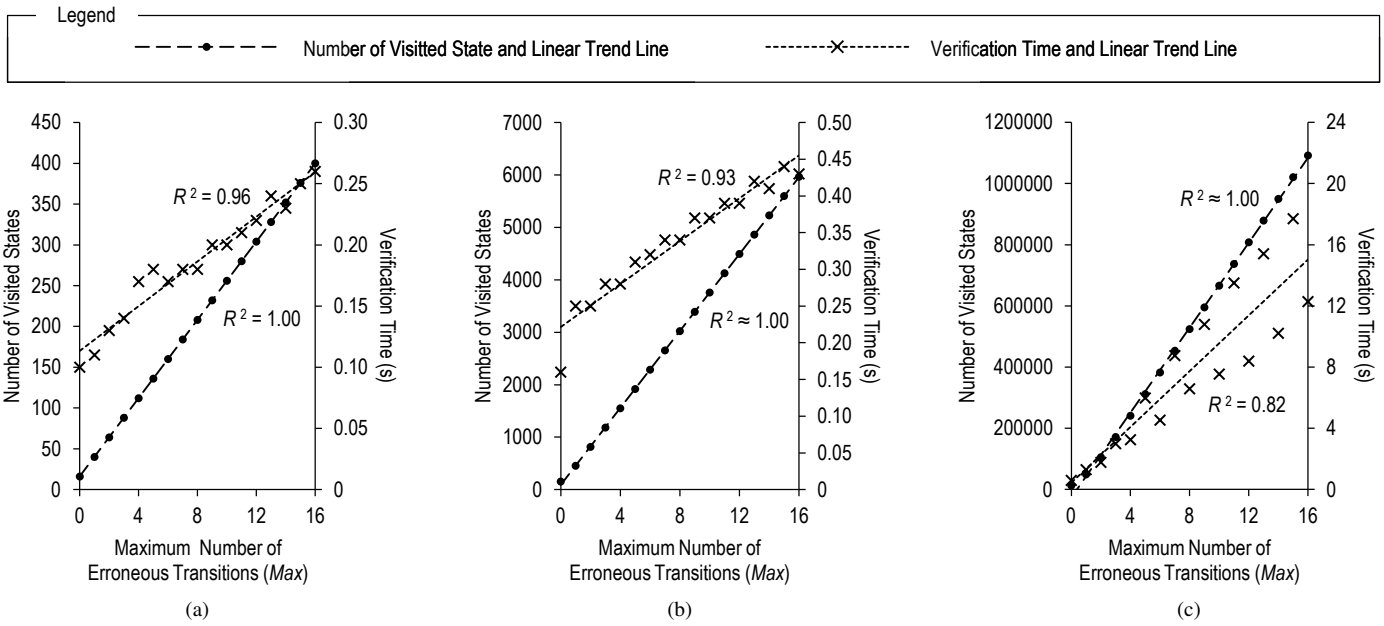


Fig. 5. Plot of the verification results (verification time in seconds and the number of visited states) for maximum numbers of erroneous transitions (*Max*) between 0 and 16 for each of the three task structures (a), (b), and (c) from Fig. 3 presented here in (a), (b), and (c) respectively. In all plots, the number of visited states is reported on the left y-axis and verification time is reported on the right y-axis. Linear trends lines are presented for each plot with their corresponding R^2 statistic. The $\approx$ symbol is used to indicate when the R^2 statistic rounded up to 1.

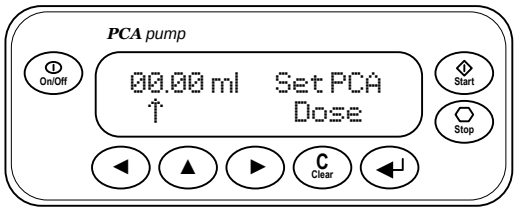


Fig. 6. The PCA pump HDI for programming prescriptions.

system model contained sub-models representing the practitioner's mission, the HDI, the device automation, and human task behavior automatically translated from instantiated EOFM task models using both their normative representation and the erroneous human behavior generation method presented above.

1) Human mission: The practitioner's mission was to program prescriptions into the pump. The prescription was represented by three values: a PCA dosage (*PrescribedPCADose*), a minimum delay between dosages (*PrescribedDelay*), and a limit on the total dosage delivered in an hour (*PrescribedLimit*). To control the complexity of the model, all values (including those in prescriptions) were represented abstractly as either being *Correct* or *Incorrect*. Every value in a prescription was always *Correct* since these were the values the practitioner was attempting to program into the pump.

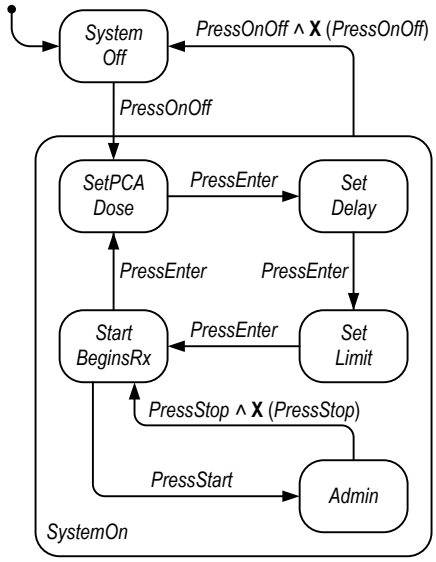
2) Human-device interface: The HDI represented the state of the LCD (*Display*) which indicated when the system was off (*SystemOff*), when the dosage could be programmed (*SetPCADose*), when the delay could be programmed (*SetDelay*), when the one hour limit could be programmed (*SetLimit*), when prescription delivery could be started or reviewed (*StartBeginsRx*), and when treatment was being administered (*Admin*). It would also display the value (*Correct* or *Incorrect*) associated with the *SetPCADose*, *SetDelay*, and *SetLimit* states. It received human action inputs from the eight buttons: *PressOnOff*, *PressStart*, *PressStop*, *PressLeft*, *PressUp*, *PressRight*, *PressClear*, and *PressEnter*.

3) Device automation: The model of the device automation controlled the interface states (Fig. 7(a)) and displayed values (Fig. 7(b)) based on internal variables and human actions.

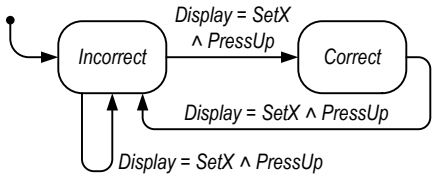
4) Human task behavior: An instantiated EOFM was created encompassing the following high-level goal directed behaviors for normatively performing activities with the pump (Fig. 8): (a) turning on the pump, (b) stopping the infusion of medication, (c) selecting whether to start or review an entered prescription, (d) turning off the pump, and (e) entering prescribed values (PCA dosages, delays, and one hour limits).

The tasks most relevant to this discussion are those related to the programming of prescription values, all of which have the form shown in Fig. 8(e). For a given value *X*, the corresponding EOFM becomes relevant when the interface for setting that value is displayed. A practitioner first changes the displayed value to match that from the prescription. The value can be changed by selecting different digits with Left and Right button presses (*PressLeft* and *PressRight*), clearing the display by pressing the Clear button (*PressClear*), or changing a digit by pressing the Up button (*PressUp*). The practitioner repeats the change activity (a repeat condition) as long as the displayed value does not match the prescription value. The displayed value is accepted by pressing the enter key.

The EOFM instance (Fig. 8) was translated twice into SAL code and incorporated into the larger formal system model: once for normative behavior ($Max = 0$) and once for erroneous human behavior with a maximum of one erroneous transition ($Max = 1$). The normative behavior model's EOFM representation was 147 lines of code. Its corresponding formal,



(a) *Display*



(b) *ValueX*

Fig. 7. State transition model representation of the formal model of the PCA pump's automation. Rounded rectangles represent states. Arrows indicate guarded transitions between states. Transitions are labeled with transition logic. Note that variables in transition logic with the *Press* prefix indicate that a human has pressed a button on the HDI. (a) The state of LCD display. (b) The behavior used to model the state of prescription value *X*, where *X* can be any value associated with a prescription: *PCADose*, *Delay*, *Limit*.

normative representation was 475 lines of SAL code. The erroneous behavior model was 784 lines of SAL code.

B. Specification and Verification

We use linear temporal logic [67] to construct the specification in (3)[4]. This asserts that, when treatment is administering, the entered prescription should always match the one from the mission.

$$\mathbf{G}\left(\begin{array}{l}(Display = Admin)\\ \Rightarrow \left(\begin{array}{l} ValuePCADose = PrescribedPCADose\\ \wedge\ ValueDelay = PrescribedDelay\\ \wedge\ ValueLimit = PrescribedLimit\end{array}\right)\end{array}\right) \quad (3)$$

When checked against the formal system model with the translated normative task behavior model, it verified to true in 2 minutes and 46 seconds having visited 4,072,083 states.

The formal system model containing the erroneous human behavior model produced a counterexample after 1 minute and 10 seconds after visiting 1,591,373 states. We used our visualizer [64] to diagnose the discovered problem. This revealed the following failure sequence:

[4]All models were also checked to ensure that any confirmation of (3) was not due to vacuous truth.

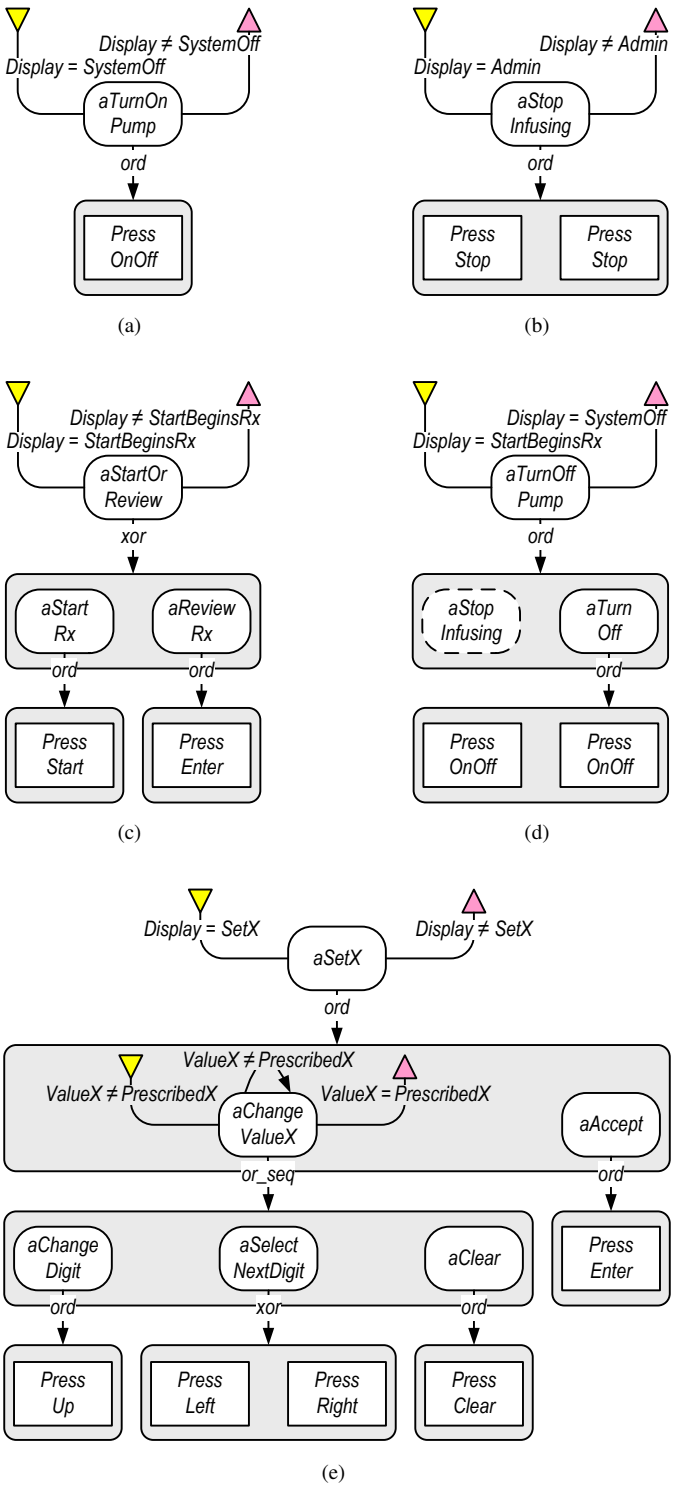


Fig. 8. Visualization of the instantiated EOFM for normatively programming prescriptions into the PCA pump. Note that (e) represents a generic pattern for programming value *X* into the pump where *X* can be *PCADose*, *Delay*, or *Limit*. Also note that the dotted line around *aStopInfusing* in (d) indicates that *aTurnOffPump* is referencing the activity *aStopInfusing* defined in (b).

1) The pump started in the off state and the practitioner had to program in a prescription specifying a PCA dose, a delay, and a one hour limit.
2) The practitioner turned the pump on by pressing the on/off button, putting the pump's interface in the PCA

dosage programming state (*SetPCADose*) with a displayed value of *Incorrect*.
3) The practitioner pressed the up button until the value was *Correct*.
4) The practitioner accepted the PCA dosage by pressing the enter button, causing the pump's interface to transition to the delay programming state (*SetDelay*) with a displayed value of *Incorrect*.
5) Rather than perform the activity for changing the delay value until the value was *Correct*, the practitioner erroneously omitted the *aChangeValueDelay* activity (see Fig. 8(e) with *X* = *Delay*), an erroneous *Ready* to *Done* transition.
6) The practitioner accepted the *Incorrect* delay by pressing the enter button, causing the pump's interface to transition to the one hour limit programming state (*SetLimit*) with a displayed value of *Incorrect*.
7) The practitioner pressed the up button until the value was *Correct*.
8) The practitioner accepted the one hour limit by pressing the enter button, causing the pump's interface to transition to the state for reviewing or starting treatment (*StartBeginsRx*).
9) The practitioner started treatment by pressing the start button, causing treatment to administer (*TreatmentAdministering*). Thus, the specification had been violated with the pump administering treatment with an unprescribed delay value.

C. Addressing the Discovered Problem

We can use our method to investigate potential interventions that mitigate the discovered problem. One possibility is to have the practitioner review the entered prescription every time it changes. One way this can be accomplished is by making changes to the device automation. For example, the pump could be modified so that it keeps track of whether or not the practitioner has reviewed the entered prescription or not and will only let her start the administration of treatment after an entered or changed prescription has been reviewed.

If we make this change to the PCA pump model, we can re-run the verification against (3) with the erroneous human behavior model from above. When we did this, the specification verified to true in 23 minutes and 41 seconds having visited 43,033,617 states.

While this solution will work, it may not always be viable. For example, if the analyses are being conducted by a hospital using the pump in question, the analysts may not be able to actually modify the device automation. However, they may be able to change the procedures and training. In such a situation, the hospital could require that practitioners always review a new or changed prescription once before administering treatment.

To implement this in our models, we made changes to the instantiated EOFM. We introduced a Boolean local variable called *Reviewed* to the instantiated EOFM with an initial value of false to allow the practitioner to keep track of whether or not (true or false respectively) she has reviewed a new or modified prescription. Then we modified the task structures (originally from Fig. 8) so that *Reviewed*'s value is appropriately updated when programming in a prescription (see Fig. 9): *Reviewed* is set to false when the pump is turned on (Fig. 9(a)); *Reviewed* is set to true when the practitioner chooses to review a prescription, and preconditions are added so that a prescription is administered if *Reviewed* is true and reviewed if it is false (Fig. 9(b)); and *Reviewed* is set to false every time the practitioner makes a change to a prescription value (Fig. 9(c)).

The translator was re-run with the modified tasks and *Max* = 1, and the translated tasks were paired with the original mission, HDI, and automation models. When (3) was checked against this system model, it verified to true in 30 minutes and 14 seconds having visited 44,532,648 states.

D. Multiple Erroneous Behaviors

To see how robust our solutions were to more than one attention failure, we rechecked the last two models (the one with the modified device automation and the one with the modified task behavior) with *Max* = 2 against the specification in (3). In both cases, the formal verifications failed. The model with the modified device automation produced a counterexample after 32 minutes and 19 seconds after visiting 98,736,453 states. The model with the modified task behavior produced one in 1 minute and 48 seconds having visited 3,655,441 states.

An examination of these results using our visualizer [64] revealed that in both counterexamples a somewhat unrealistic erroneous behavior was contributing to the observed failures. In both cases, when it would have been appropriate for the practitioner to program the delay into the pump (the task in Figs. 8(e) and 9(c) with *X* = *Delay*), the practitioner made a commission (an erroneous *Ready* to *Executing* transition) with the activity for starting or reviewing a prescription (*aStartOrReview* from Figs. 8(c) and 9(b)). This resulted in the practitioner pressing the enter button (via the *aStartRx* activity) without making a change to the delay value and, for the model with the modified task behavior, setting *Reviewed* to true. Also in both examples, this erroneous behavior interacted with another erroneous behavior to ultimately produce the discovered failure. In the model with the modified device automation, the practitioner made an omission (he did not change/correct the delay value) when reviewing the entered delay, thus allowing an incorrect delay to be administered. In the model with the modified task behavior, after erroneously performing *aStartOrReview* (pressing enter and setting *Reviewed* to true), the practitioner made an omission when programming in the one hour limit (an erroneous *Ready* to *Done* transition for *aSetLimit* from Fig. 9(c) with *X* = *Limit*). Thus, *Review* was true when it came time for the practitioner to choose between starting or reviewing the prescription (Fig. 9(b)), resulting in the practitioner pressing start which administered a prescription containing incorrect delay and limit values.

While theoretically possible, these failure sequences are fairly unrealistic: it is unlikely that a practitioner would mistake the interface state for programming in the delay for the one for starting or reviewing a prescription. Thus, we used

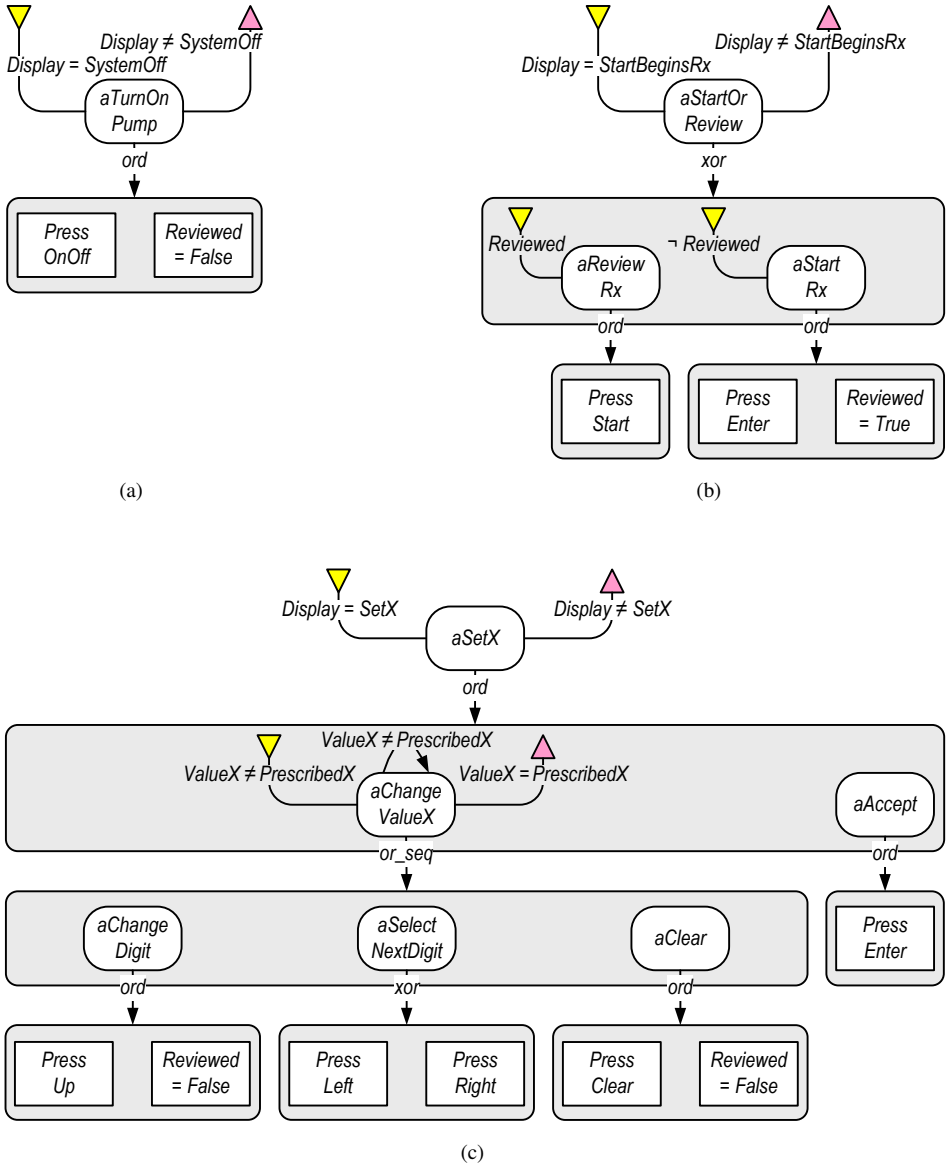


Fig. 9. Visualization of the modified tasks from Fig. 8 to enable the practitioner to remember if he has reviewed a new or modified prescription before administering treatment based on the value of local variable called *Reviewed*. Note that the Boolean local variable *Reviewed* is assigned values as a mental action at the bottom of the task model hierarchy. (a) The updated version of the task from Fig. 8(a). (b) The updated version of the task from Fig. 8(c). (c) The updated version of the task pattern from Fig. 8(e).

our technique for eliminating undesirable erroneous behaviors to remove the erroneous *Ready* to *Executing* transition of *aStartOrReview* (Figs. 8(c) and 9(b)) from both models. With these changes made, both models were again checked against the specification in (3). Both verifications produced counterexamples.

The model with the modified device automation produced a counterexample in 24 minutes having visited 85,868,280 states. This revealed that the practitioner made the same error twice: when both initially programming and reviewing the prescription, the practitioner erroneously omitted the *aChangeDelayValue* activity (from the task in Fig. 8(e) with *X = Delay*). Thus, the practitioner entered the incorrect delay and failed to correct during the review process. This resulted in a prescription with an incorrect delay being administered.

The model with the modified task behavior produced a counterexample in 1 minute and 55 seconds having visited 5,478,564 states. When this counterexample was visualized, it revealed that, when initially programming the prescription into the pump, the practitioner erroneously omitted the *ChangeDelayValue* activity (from the task in Fig. 9(c) with *X = Delay*). Then, when the practitioner was asked whether to review or administer treatment, the practitioner made a commission: when *aStartOrReview* (Fig. 9(b)) was executing, the *aStartRx* erroneously transitioned from *Ready* to *Executing*. This also resulted in the practitioner administering a prescription with an incorrect delay.

These two failures are much more plausible than the ones found before removing *aStartOrReview*'s erroneous *Ready* to *Executing* transition. Unfortunately, it is not immediately clear how the device automation or human training could be modified to prevent these or similar failures from occurring.

VI. Discussion

The method presented here makes a novel contribution to model-driven design and analysis techniques that use formal methods to evaluate the role of human behavior in system safety. By using task analytic behavior models, erroneous behavior generation, formal modeling, and model checking, the presented method gives analysts the ability to use task analytic human behavior models to evaluate if the modeled human behavior will or will not result in violations of system safety. Further, by adding erroneous transitions to the formal semantics of an EOFM's activity execution state, each representing erroneous applications of strategic knowledge (pre, repeat, and completion conditions), we are capable of automatically including the observable manifestation of attentional failures associated with Reason's [54] slips (omission, repetition, or commission) in our formal verification analyses. The number of possible erroneous transitions is constrained by a maximum and a counter preventing generated erroneous behaviors from making the task behavior model unbounded. Thus, analysts can use our method to determine if their system is safe for up to the maximum number of potentially unanticipated erroneous behaviors associated with attentional failures (erroneous transitions).

The PCA pump application illustrates how this method can be used to evaluate a safety critical system that depends on HAI. Employing this example we used our method to demonstrate how one can show that a device is safe when the human operator behaves normatively. We then showed how a violation of system safety could be discovered using the presented erroneous behavior generation technique. The method was then used to explore different design or training interventions that could be used to correct the discovered problem. Finally, we increased the number of erroneous behaviors to assess how robust our designs were to additional human operator attentional failures. After pruning our results (using the method's process for doing), we showed that the presented interventions were not robust for up to a maximum of two attentional failures.

While the method has shown itself to have utility, there are many directions for future research.

A. Comparison with Other Task-based Approaches

When compared to the other techniques that use task analytic models to evaluate the impact of erroneous human behavior on system safety, our method has several advantages. The vast majority of previous techniques have focussed on verifying system safety with normative human task behavior, and do not consider erroneous human behavior [37]–[41], [43]–[46], [48]–[50]. Those that do include erroneous behavior, have typically focussed on manually inserting them into task analytic models at locations the analysts think might cause problems [47], [58]–[62]. Thus, our method is advantageous in that it allows analysts to evaluate the impact of erroneous human behaviors they might not anticipate will cause problems.

The presented method also has an advantage over the automated approach developed by Bolton et al. [51] that focussed on generating erroneous human behaviors using Hollnagel's [3] zero-order phenotypes of erroneous action. While the method presented in [51] works well for evaluating the impact of small numbers of erroneous phenotypical behaviors on system safety, it does a poor job of replicating higher order attentional failures like those explored by Paternò and Santoro [59]. However, our new method is capable of generating these types of higher order failures without considering all of the complex combinations of extraneous actions that would be required to generate similarly ordered erroneous behaviors using the technique from [51]. It is important to note that because these two generation techniques produce different erroneous behaviors, there may be advantages to using them synergistically. The method in [51] can be used to generate lower level erroneous acts and can be used to generate many more extraneous behaviors while the method presented in this paper could be used to explore higher order erroneous behaviors based on attentional failures. Future work should investigate this possibility.

B. Comparison with Other Methods

Bolton et al. [51] discuss several other techniques that allow erroneous human behaviors to be considered in formal verification analyses that do not make use of task models. These include techniques that only use a model of the HDI [8]–[11], [17]–[20], methods that use human mental models as part of larger system models [14]–[16], [21]–[23], and approaches that use cognitive architectures as part of the system model [24]–[34]. Bolton et al. note that there are tradeoffs between these and the task model-based approaches. Techniques that only use HDI models can find any possible failure sequence and are often more scalable than other approaches, but provide little insights into why an erroneous behavior occurs and may not be suitable for evaluating systems (such as aircraft or medical devices) where design interventions cannot eliminate all potential system problems. Approaches that use mental models are particularly good at finding system conditions that could produce mode confusion, but do not explicitly model the impact of erroneous behaviors. Methods that use cognitive architectures explicitly model the cognitive mechanisms behind erroneous behavior and thus provide insights into why a problematic erroneous behavior can occur. However, these methods require that the cognitive mechanisms for the erroneous behavior be explicitly incorporated into the model and use modeling approaches that are not commonly employed by the human factors and systems engineering communities. Finally, task model-based analyses (like the one described in this paper) also provide insights into why an erroneous behavior occurs, but make use of human behavior models that systems engineers more commonly use. However, they may not scale as well as other approaches.

There may be utility in developing a framework that could support all of these methods. In such a framework, an analyst could deploy each of the techniques where she felt it was appropriate. Future work should investigate how these techniques could be incorporated into an integrated framework with heuristics for guiding analysts towards the methods that are most appropriate for the system they are evaluating.

C. Scalability

A significant increase in the statespace size and verification times was observed between the normative human behavior model (4,072,083 states evaluated in 2 minutes and 46 seconds) and the one with a maximum of 1 erroneous transition (43,033,617 states evaluated in 23 minutes and 41 seconds). Such increases are likely to limit the applicability of the presented method. However the benchmark results indicate that the method scales linearly with the maximum number of erroneous transitions. Thus, the method presented here scales much better than the phenotypical erroneous behavior generation method in [51] which scaled exponentially with an increase in the maximum number of allowable erroneous actions. Thus, the method presented here would likely be applicable to more complex systems.

Despite this advantage, improvements in scalability would still increase the applicability of the method. The EOFM to SAL translation process includes all of the intermediary transitions associated with the execution state of activities (Figs. 1 and 2). It is conceivable that the execution state of each activity could be represented exclusively in terms of the execution states of actions. Additionally, the mechanisms that implement the coordination protocol used to compose the translated human task behavior model with the other models in the formal system model [43] add to the statespace. More efficient means of achieving the desired behavior may exist. Alternative modeling architectures, like those based on synchronous observers (see [42]), may prove to be more statespace-efficient. Future work should investigate these and other methods for potentially improving the scalability of the presented method.

D. Method Extensions

The method presented here only depends on the interpretation of activity level strategic knowledge. Thus, although the method is capable of generating omission, repetition, or commission that can result from slips related to strategic knowledge, it is not capable of replicating ordering errors (a type of commission) for activities contained in an *ord* decomposition or other violations of task execution order encoded into activities' and actions' start, end, and reset conditions. Future work should investigate how to accomplish this. Additionally, the method only addresses erroneous human behaviors that can be replicated with an erroneous transition between execution states (Fig. 2). However, there may be erroneous transitions associated with a non-transition (a lack transition between execution states when there should be one). Specifically, an omission could occur if the human operator doesn't properly attend to when an activity should transition between *Ready* and *Executing*, and stay in the *Ready* state. Future work should investigate how to replicate this behavior in our method.

EOFM formal semantics do not allow for task models to be abandoned or resumed. This is problematic because erroneous transitions can lead to task deadlock (a case where the task cannot continue executing) which is unrealistic. Real human operators may attempt to abandon, resume, or restart tasks the system will not let them perform. Future work should investigate how to incorporate this behavior into EOFM.

The erroneous behavior generation process discussed here is only capable of replicating capture slips for activities in a particular peer group: either within a given decomposition, or all parent level activities. However, capture slips can also manifest as a human operator performs all or part of a completely unrelated activity, especially if the activities occur under similar circumstances or are composed of similar sequences of behavior [54]. Future work should investigate how such slips could be incorporated into our erroneous behavior generation process.

Reason's Generic Error Modeling System [54] classifies erroneous behaviors beyond the slip designations that have been discussed here. Specifically, slips only relate to erroneous behaviors that occur as a result of attentional failures that cause the human operators to incorrectly perform tasks that they know how to perform correctly. A different class of erroneous behaviors, mistakes, occur when the human operator intentionally performs an erroneous behavior because he does not know how to perform a task correctly. This can occur either because of rule-based or knowledge-based failures. Rule based mistakes occur when the human performs a valid rule or schema for achieving a goal in an incorrect place or performs an invalid rule. Failures at the knowledge level occur when the human operator has incorrect knowledge about the domain or environment. Future work should investigate how to generate mistakes as part of our infrastructure.

E. Use in Design

The various analyses that were presented demonstrate how the method can be used to evaluate different designs or system conditions: normative, erroneous, and modified human task behavior models were evaluated as well as two different implementations of the device automation. However, as was observed in the analyses with a maximum of two erroneous transitions, it is not always clear how counterexample results can be used to influence design. In fact, all model checking analyses suffer from this problem. Since a counterexample only shows a single path of failure, and a model checker will always produce the same counterexample for the same input model and specification, it may not be clear how to modify the design of a device to not only correct the discovered problem but also correct all problems of a similar nature or form without introducing new problems.

For example, in the application presented in this paper, the model checker found a counterexample in which the practitioner performed an omission when programming a delay into the PCA pump. However, there are likely other erroneous behaviors that could potentially result in a practitioner programming in an incorrect prescription (e.g., performing an omission when programming in a PCA dosage or one hour limit). Thus, to find other possible failure scenarios, the analyst would need to change the analysis by removing the specific omission from consideration. One way of accomplishing this would be to remove the contributory erroneous transition from the translated version of the instantiated EOFM using the

approach described in section III-E (in the SAL code, this would be as simple as commenting out the erroneous *Ready* to *Done* transition associated with *aChangeValueDelay*). Using this approach (iteratively removing contributory erroneous transitions), an analyst could discover all of the potential erroneous transitions that could result in a property violation.

However, even with such insights, an analyst may have trouble comparing the information contained in multiple counterexamples and synthesizing it into design interventions. Thus, advances in counterexample visualizations [64] and other formal modeling decision aids may help analysts perform this task. Future work should investigate how these technologies could be used to help analysts design model-checking-discovered problems out of their systems.

ACKNOWLEDGMENT

The project described was supported in part by Grant Number T15LM009462 from the National Library of Medicine (NLM), NASA Cooperative Agreement NCC1002043, and NASA award NNA10DE79C. 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.

REFERENCES

[1] T. B. Sheridan and R. Parasuraman, "Human-automation interaction," *Reviews of human factors and ergonomics*, vol. 1, no. 1, pp. 89–129, 2005.

[2] R. Parasuraman, T. Sheridan, and C. Wickens, "A model for types and levels of human interaction with automation," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 30, no. 3, pp. 286–297, 2000.

[3] E. Hollnagel, "The phenotype of erroneous actions," *International Journal of Man-Machine Studies*, vol. 39, no. 1, pp. 1–32, 1993.

[4] H. Hussmann, G. Meixner, and Z. Detlef, *Model-Driven Development of Advanced User Interfaces*. Berlin: Springer, 2011.

[5] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using formal verification to evaluate human-automation interaction, a review," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, in press.

[6] J. M. Wing, "A specifier's introduction to formal methods," *Computer*, vol. 23, no. 9, pp. 8, 10–22, 24, 1990.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. Cambridge: MIT Press, 1999.

[8] J. C. Campos and M. D. Harrison, "Systematic analysis of control panel interfaces using formal tools," in *Proceedings of the 15th International Workshop on the Design, Verification and Specification of Interactive Systems*. Berlin: Springer, 2008, pp. 72–85.

[9] G. D. Abowd, H. Wang, and A. F. Monk, "A formal technique for automated dialogue development," in *Proceedings of the 1st Conference on Designing Interactive Systems*. New York: ACM, 1995, pp. 219–226.

[10] J. C. Campos and M. D. Harrison, "Formally verifying interactive systems: A review," in *Proceedings of the Fouth International Eurographics Workshop on the Design, Specification, and Verification of Interactive Systems*. Berlin: Springer, 1997, pp. 109–124.

[11] M. Thomas, "The story of the therac-25 in lotos," *High Integrity Systems*, vol. 1, no. 1, pp. 3–15, 1994.

[12] D. A. Norman, "The problem with automation: Inappropriate feedback and interaction, not over-automation," *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, vol. 327, pp. 585–593, 1990.

[13] N. B. Sarter and D. D. Woods, "How in the world did we ever get into that mode? Mode error and awareness in supervisory control," *Human Factors*, vol. 37, no. 1, pp. 5–19, 1995.

[14] B. Buth, "Analyzing mode confusion: An approach using FDR2," in *Proceeding of the 23rd International Conference on Computer Safety, Reliability, and Security*. Berlin: Springer, 2004, pp. 101–114.

[15] A. Degani and M. Heymann, "Formal verification of human-automation interaction," *Human Factors*, vol. 44, no. 1, pp. 28–43, 2002.

[16] J. Rushby, "Using model checking to help discover mode confusions and other automation surprises," *Reliability Engineering and System Safety*, vol. 75, no. 2, pp. 167–177, 2002.

[17] J. C. Campos and M. D. Harrison, "Model checking interactor specifications," *Automated Software Engineering*, vol. 8, no. 3, pp. 275–310, 2001.

[18] A. Joshi, S. P. Miller, and M. P. Heimdahl, "Mode confusion analysis of a flight guidance system using formal methods," in *Proceedings of the 22nd Digital Avionics Systems Conference*. Piscataway: IEEE, October 2003, pp. 2.D.1-1–2.D.1-12.

[19] N. G. Leveson, L. D. Pinnel, S. D. Sandys, S. K., and J. D. Reese, "Analyzing software specifications for mode confusion potential," in *Proceedings of the Workshop on Human Error and System Development*. Glasgow: University of Glasgow, 1997, pp. CD–ROM.

[20] J. C. Campos and M. D. Harrison, "Modelling and analysing the interactive behaviour of an infusion pump," in *Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems*. Potsdam: EASST, 2011.

[21] J. Bredereke and A. Lankenau, "Safety-relevant mode confusions–modelling and reducing them," *Reliability Engineering and System Safety*, vol. 88, no. 3, pp. 229–245, 2005.

[22] D. Javaux, "A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system," *Reliability Engineering and System Safety*, vol. 75, pp. 147–165, 2002.

[23] P. H. Wheeler, "Aspects of automation mode confusion," Master's thesis, Massachusetts Institute of Technology, Cambridge, 2007.

[24] A. Blandford, R. Butterworth, and J. Good, "Users as rational interacting agents: Formalising assumptions about cognition and interaction," in *Proceedings of the 4th International Eurographics Workshop, on the Design, Specification and Verification of Interactive Systems*, vol. 97. Berlin: Springer, 1997, pp. 45–60.

[25] A. Blandford, R. Butterworth, and P. Curzon, "Models of interactive systems: A case study on programmable user modelling," *International Journal of Human-Computer Studies*, vol. 60, no. 2, pp. 149–200, 2004.

[26] R. Butterworth, A. Blandford, and D. Duke, "Demonstrating the cognitive plausibility of interactive system specifications," *Formal Aspects of Computing*, vol. 12, no. 4, pp. 237–259, 2000.

[27] ——, "The role of formal proof in modelling interactive behaviour," in *Proceedings of the 5th International Eurographics Workshop on the Design, Specification and Verification of Interactive Systems*. Berlin: Springer, 1998, pp. 87–101.

[28] P. Curzon and A. Blandford, "From a formal user model to design rules," in *Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification*. London: Springer, 2002, pp. 1–15.

[29] ——, "Formally justifying user-centered design rules: A case study on post-completion errors," in *Proceedings of the 4th International Conference on Integrated Formal Methods*. Berlin: Springer, 2004, pp. 461–480.

[30] P. Curzon, R. Rukšėnas, and A. Blandford, "An approach to formal verification of humancomputer interaction," *Formal Aspects of Computing*, vol. 19, no. 4, pp. 513–550, 2007.

[31] R. Rukšėnas, P. Curzon, J. Back, and A. Blandford, "Formal modelling of cognitive interpretation," in *Proceedings of the 13th International Workshop on the Design, Specification, and Verification of Interactive Systems*. London: Springer, 2007, pp. 123–136.

[32] R. Rukšėnas, J. Back, P. Curzon, and A. Blandford, "Formal modelling of salience and cognitive load," in *Proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems*. Amsterdam: Elsevier Science Publishers, 2008, pp. 57–75.

[33] R. Rukšenas, J. Back, P. Curzon, and A. Blandford, "Verification-guided modelling of salience and cognitive load," *Formal Aspects of Computing*, vol. 21, no. 6, pp. 541–569, 2009.

[34] T. A. Basuki, A. Cerone, A. Griesmayer, and R. Schlatte, "Model-checking user behaviour using interacting components," *Formal Aspects of Computing*, pp. 1–18, 2009.

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

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

[37] S. Basnyat, P. Palanque, B. Schupp, and P. Wright, "Formal socio-technical barrier modelling for safety-critical interactive systems design," *Safety Science*, vol. 45, no. 5, pp. 545–565, 2007.

[38] S. Basnyat, P. Palanque, R. Bernhaupt, and E. Poupart, "Formal modelling of incidents and accidents as a means for enriching training material for satellite control operations," in *Proceedings of the Joint*

ESREL 2008 and 17th SRA-Europe Conference. London: Taylor and Francis Group, 2008, pp. CD–ROM.
[39] J. C. Campos, "Using task knowledge to guide interactor specifications analysis," in *In Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification.* Berlin: Springer, 2003, pp. 171–186.
[40] Y. Aït-Ameur, M. Baron, and P. Girard, "Formal validation of HCI user tasks," in *Proceedings of the International Conference on Software Engineering Research and Practice*. Las Vegas: CSREA Press, 2003, pp. 732–738.
[41] Y. Aït-Ameur and M. Baron, "Formal and experimental validation approaches in HCI systems design based on a shared event B model," *International Journal on Software Tools for Technology Transfer*, vol. 8, no. 6, pp. 547–563, 2006.
[42] E. J. Bass, M. L. Bolton, K. Feigh, D. Griffith, E. Gunter, W. Mansky, and J. Rushby, "Toward a multi-method approach to formalizing human-automation interaction and human-human communications," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. Piscataway: IEEE, 2011, pp. 1817–1824.
[43] M. L. Bolton and E. J. Bass, "Formally verifying human-automation interaction as part of a system model: Limitations and tradeoffs," *Innovations in Systems and Software Engineering: A NASA Journal*, vol. 6, no. 3, pp. 219–231, 2010.
[44] ——, "A method for the formal verification of human interactive systems," in *Proceedings of the 53rd Annual Meeting of the Human Factors and Ergonomics Society*. Santa Monica: HFES, 2009, pp. 764–768.
[45] M. L. Bolton, R. I. Siminiceanu, and E. J. Bass, "A systematic approach to model checking human-automation interaction using task-analytic models," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 41, no. 5, pp. 961–976, 2011.
[46] M. L. Bolton and E. J. Bass, "Using model checking to explore checklist-guided pilot behavior," *International Journal of Aviation Psychology*, vol. 22, no. 4, pp. 343–366, 2012.
[47] R. E. Fields, "Analysis of erroneous actions in the design of critical systems," Ph.D. dissertation, University of York, York, 2001.
[48] F. Paternò, C. Santoro, and S. Tahmassebi, "Formal model for cooperative tasks: Concepts and an application for en-route air traffic control," in *Proceedings of the 5th International Conference on the Design, Specification, and Verification of Interactive Systems*. Vienna: Springer, 1998, pp. 71–86.
[49] F. Paternò and C. Santoro, "Integrating model checking and HCI tools to help designers verify user interface properties," in *Proceedings of the 7th International Workshop on the Design, Specification, and Verification of Interactive Systems*. Berlin: Springer, 2001, pp. 135–150.
[50] P. Palanque, R. Bastide, and V. Senges, "Validating interactive system design through the verification of formal task and system models," in *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*. London: Chapman and Hall, Ltd., 1996, pp. 189–212.
[51] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Generating phenotypical erroneous human behavior to evaluate human-automation interaction using model checking," *International Journal of Human-Computer Studies*, vol. 70, no. 11, pp. 888–906, 2012.
[52] P. M. Jones, "Human error and its amelioration," in *Handbook of systems engineering and management*. Malden: Wiley, 1997, pp. 687–702.
[53] G. Baxter and E. Bass, "Human error revisited: Some lessons for situation awareness," in *Proceedings of the Fourth Annual Symposium on Human Interaction with Complex Systems*. Piscataway: IEEE, 1998, pp. 81–87.
[54] J. Reason, *Human Error*. New York: Cambridge University Press, 1990.
[55] F. Paternò, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models," in *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*. London: Chapman and Hall, Ltd., 1997, pp. 362–369.
[56] C. M. Mitchell and R. A. Miller, "A discrete control model of operator function: A methodology for information display design," *IEEE Transactions on Systems Man Cybernetics Part A: Systems and Humans*, vol. 16, no. 3, pp. 343–357, 1986.
[57] E. J. Bass, S. T. Ernst-Fortin, R. L. Small, and J. Hogans, "Architecture and development environment of a knowledge-based monitor that facilitate incremental knowledge-base development," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 34, no. 4, pp. 441–449, 2004.
[58] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using formal methods to predict human error and system failures," in *Proceedings of the 2nd International Conference on Applied Human Factors and Ergonomics*. Las Vegas: Applied Human Factors and Ergonomics International, 2008, pp. CD–ROM.
[59] F. Paternò and C. Santoro, "Preventing user errors by systematic analysis of deviations from the system task model," *International Journal of Human-Computer Studies*, vol. 56, no. 2, pp. 225–245, 2002.
[60] R. Bastide and S. Basnyat, "Error patterns: Systematic investigation of deviations in task models," in *Task Models and Diagrams for Users Interface Design*. Berlin: Springer, 2007, pp. 109–121.
[61] S. Basnyat and P. Palanque, "A task pattern approach to incorporate user deviation in task models," in *Proceedings of the first ADVISES Young Researchers Workshop*. Roskilde: Risϕ National Laboratory, 2005, pp. 10–19.
[62] P. Palanque and S. Basnyat, "Task patterns for taking into account in an efficient and systematic way both standard and erroneous user behaviours," in *IFIP 13.5 Working Conference on Human Error, Safety and Systems Development*. Norwell: Kluwer Academic Publisher, 2004, pp. 109–130.
[63] D. A. Thurman, A. R. Chappell, and C. M. Mitchell, "An enhanced architecture for OFMspert: A domain-independent system for intent inferencing," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. Piscataway: IEEE, 1998, pp. 955–960.
[64] M. L. Bolton and E. J. Bass, "Using task analytic models to visualize model checker counterexamples," in *Proceedings of the 2010 IEEE International Conference on Systems, Man, and Cybernetics*. Piscataway: IEEE, 2010, pp. 2069–2074.
[65] L. De Moura, S. Owre, and N. Shankar, "The SAL language manual," Computer Science Laboratory, SRI International, Menlo Park, Tech. Rep. CSL-01-01, 2003.
[66] M. L. Bolton, "Automatic validation and failure diagnosis of human-device interfaces using task analytic models and model checking," *Computational and Mathematical Organization Theory*, pp. 1–25, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10588-012-9138-6
[67] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, A. R. Meyer, M. Nivat, M. Paterson, and D. Perrin, Eds. Cambridge: MIT Press, 1990, ch. 16, pp. 995–1072.

Matthew L. Bolton (S'05-M'10) received the B.S. in computer science in 2003, the M.S. in systems engineering in 2006, and the Ph.D. in systems engineering in 2010 from the University of Virginia, Charlottesville, USA.

He is an Assistant Professor of industrial engineering in the Department of Mechanical and Industrial Engineering at the University of Illinois at Chicago. His research is primarily focused on the development of tools and techniques for using human performance modeling, task analysis, and formal methods to analyze, design, and evaluate complex, safety-critical systems.

Ellen J. Bass (M'98-SM'03) received the B.S. Eng. and B.S. Econ. degrees from the University of Pennsylvania, Philadelphia, the M.S. degree from the State University of New York at Binghamton, and the Ph.D. degree from the Georgia Institute of Technology, Atlanta.

She is a Professor in the College of Information Science and Technology and the College of Nursing and Health Professions at Drexel University. She has 30 years of industry and research experience in human-centered systems engineering in the domains of air transportation, meteorology, healthcare and informatics. The focus of her research is to develop theories of human performance, quantitative modeling methodologies, and associated experimental designs that can be used to evaluate human-automation interaction in the context of total system performance. The outcomes of the research can be used in the systems engineering process: to inform system requirements, procedures, display designs and training interventions and to support system evaluation.