Evaluating Human-automation Interaction Using Task Analytic Behavior Models, Strategic Knowledge-based Erroneous Human Behavior Generation, and Model Checking

Matthew L. Bolton
San José State University Research Foundation
NASA Ames Research Center
Moffett Field, CA 94035
matthew.l.bolton@nasa.gov

Ellen J. Bass
Department of Systems and Information Engineering
University of Virginia
Charlottesville, VA 22904
ejb4n@virginia.edu

***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 task analytic models encompassing both erroneous and normative human behavior from normative task models by manipulating modeled strategic knowledge. Resulting models can be automatically translated 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. This method is 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 and a potential solutions is explored. 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] is particularly important to the operation of safety-critical systems. Erroneous human behavior [2], 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. However, many of these traditional techniques are not exhaustive and can miss potentially dangerous interactions.

Analysis techniques found in formal methods do not have this limitation. Formal methods are a set of languages and techniques for the modeling, specification, and verification of systems (usually computer hardware or software) [3]. Model checking is an automated approach used to verify that a formal model of a system satisfies a set of desired properties (a specification) [4]. A formal model describes a system as a set of variables and transitions between variable values (states). Specification properties are typically represented in a temporal logic (see [5]) where the variables that describe the formal system model are used to construct propositions. Verification is the process of proving that the system meets the properties in the specification. Model checking does this automatically by searching a system's entire statespace in order to determine if these criteria hold. If there is a violation, an execution trace is produced (a counterexample). This depicts a model state (a valuation of the model's variables) corresponding to a specification violation along with a list of the incremental model states that led up to it.

In this paper we present a method in which task analytic behavior models can be used to automatically generate erroneous human behavior based on the misapplication of strategic knowledge. The impact of this generated behavior can be evaluated with model checking. Herein, we first discuss erroneous human behavior taxonomies and we explore how they have been used with task analytic behavior models and formal verification to evaluate safety-critical systems. We then present our new method and illustrate it with a case study. 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 (see [6]). Of relevance to this work are Hollnegel's phenotypes of erroneous action [2] and Reason's Generic Error Modeling [7].

Hollnagel [2] classifies 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.

Reason [7] classifies erroneous behaviors based on their cognitive causes, their genotypes. As part of this, Reason identified a set 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 activity or action normatively. A person can omit an action or high level activity (which may contain multiple actions) by failing to notice that the necessary conditions for performing the activity are true, possibly due to interruption or not attending to information at the right time. A person may erroneously repeat an action or activity 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) an inappropriate action or activity either in addition to or instead of an appropriate one. 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. Evaluating the Impact of Erroneous Behavior with Task Analytic Models and Formal Methods

When designing for HAI, task analytic methods can be used to describe the normative human behaviors [8]. The resulting models represent the mental and physical activities operators use to achieve the goals with the system. These models are often hierarchical: activities decompose into other activities and, at the lowest level, atomic actions. Strategic knowledge (herein modeled as condition logic) controls when activities can execute and modifiers between activities or actions control how they execute in relation to each other. Many can be represented with discrete graph structures [9]–[11].

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 [2] into normative task behavior models [12]–[17]. Paternò and Santoro [14] presented a different approach for modeling higher order erroneous behaviors more akin to the physical manifestation of Reason's [7] 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, human device interfaces, and the operational environment [12], [18]–[22]. 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 and Bass [23] 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 a 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 [11], [23]. 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 (see [23]), 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 [14]. Such large upper bounds will generate erroneous behavior patterns analysts may not be interested in.

C. Objectives

We have developed a method that allows analysts to automatically evaluate the impact of erroneous behaviors like those discussed by Paternò and Santoro [14] without considering the many uninteresting behaviors generated using the technique from Bolton and Bass [23] with large upper bounds on the number of erroneous acts. To accomplish this, we modify the way strategic knowledge is interpreted in task analytic behavior models (represented in the Enhanced Operator Function Model (EOFM) notation [11]) in order to replicate Reason's [7] slips and evaluate their impact on system behavior using model checking. We first describe the infrastructure in which this technique was implemented. We then present our method. 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. Avenues of future work are discussed.

II. EOFM and the Formal Verification of HAI

We have developed a method [20] to evaluate HAI formally using task analytic models of human behavior. The method utilizes a formal modeling architectural framework which encompasses models of human missions (i.e. goals), human task behavior, human-device interfaces, device automation, and the operational environment [19]. With our method, human task models are created using an intermediary language called Enhanced Operator Function Model (EOFM) [11], an XML-based human task modeling language derived from the Operator Function Model (OFM) [10], [24]. EOFMs are hierarchical and heterarchical representations of goal driven activities that decompose into lower level activities, and finally, atomic actions (typically observable human actions but cognitive and perceptual actions are also possible). 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 (see examples in Fig. 4). 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, three of the nine decomposition operators [11] 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) *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 [11].

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); and its reset condition (*reset* – when it can revert to *Ready* based on the execution state of its parents). See [11] for more details.

Instantiated EOFM task models can be automatically translated [11] into the language of the Symbolic Analysis Laboratory (SAL) [25] using the language's formal semantics where they can be integrated into a larger system model using a defined architecture and coordination protocol [11], [19]. 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 [26]).

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

The erroneous human behavior generation process replicates the manifestation of Reason's [7] slips for omitting, repeating, and committing activities or actions by manipulating EOFM strategic knowledge contained in pre, repeat, and completion conditions. This was done by making changes to the EOFM's formal semantics. In this design, additional transitions (dotted arrows in Fig. 1) were added in order to describe conditions in which 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

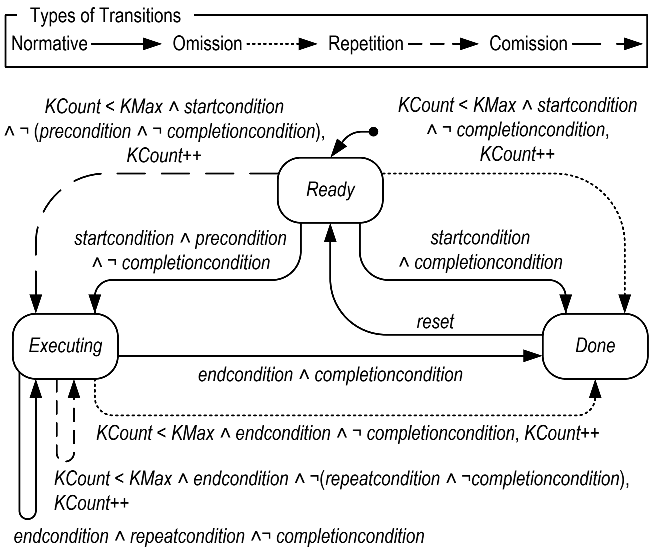

Types of Transitions
Normative
Omission
Repetition
Comission
KCount < KMax ∧ startcondition ∧ ¬ (precondition ∧ ¬ completioncondition), KCount++
KCount < KMax ∧ startcondition ∧ ¬ completioncondition, KCount++
Ready
startcondition ∧ precondition ∧ ¬ completioncondition
startcondition ∧ completioncondition
reset
Executing
Done
endcondition ∧ completioncondition
KCount < KMax ∧ endcondition ∧ ¬ completioncondition, KCount++
KCount < KMax ∧ endcondition ∧ ¬(repeatcondition ∧ ¬completioncondition), KCount++
endcondition ∧ repeatcondition ∧¬ completioncondition

Fig. 1. Modified formal semantics of an EOFM activity's execution state presented as a finite state transition system. Arrows with solid lines represent the original formal semantic transitions. Arrows with dotted lines represent additional (erroneous) transitions.

start or end condition as well as the negation of any strategic knowledge (pre, completion, or repeat condition) used by any non-erroneous transition.

Omissions can occur when an activity completes its execution too early. An activity can erroneously complete (transition from *Executing* to *Done*) if the end condition is *true* and the completion condition is not. This transition is eliminated if an activity does not have a completion condition.

Omissions can also occur when an activity fails to execute at all. An activity will erroneously not execute (transition from *Ready* to *Done*) if the start condition is *true* and the completion condition is not. This transition persists even if there is no completion condition.

An erroneous repetition occurs when an activity erroneously repeats its execution. An activity can erroneously repeat (a transition from *executing* to *executing*) if the end condition is *true* and either the repeat condition is *false* or the completion condition is *true*.

A commission occurs when an activity erroneously executes. An activity can erroneously execute (a transitions from *Ready* to *Executing*) if the start condition is *true* and either the precondition is *false* or the completion condition is *true*. This transition is only relevant if the activity has a precondition.

Too many erroneous transitions could result in an unbounded human task behavior model which would defeat the purpose of having a task model in the first place. Thus, the analyst can limit the number of erroneous transitions using an upper bound on the number of erroneous transitions (*KMax*). A variable (*KCount*) then tracks 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 (*KCount* < *KMax*). Every time an erroneous transition occurs, *KCount* is incremented by one (*KCount*++).

Our Java-based EOFM to SAL translator [11] 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 (*KMax*) as input from the analyst. KMax 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 representing the number of erroneous transitions that have occurred (*KCount*). When writing the transition logic for each activity, this implementation adds transitions (guards and variable assignments) associated with each of the dotted lines in Fig. 1[1]. The variable assignment for each erroneous transition is identical to its non-erroneous counterpart in the SAL code, with the exception that it adds the assignment which increments the erroneous transition count.

IV. Application

To illustrate how this method can be used to discover potential system problems, we present a PCA pump programming application. 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 operator (a practitioner). The HDI for the device (Fig. 2) contains a dynamic LCD and eight buttons. 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 display is used to select display information and 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. He 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.

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 as many times as they want to before treatment is administered.

A. Formal Modeling

All of the formal models were constructed using the Symbolic Analysis Laboratory (SAL) language [25]. The formal 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]This is in addition to the other, normative (un-dotted) transitions from Fig. 1 that are already produced by the translator [11]

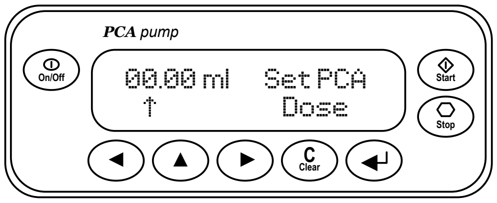


Fig. 2. The PCA pump HDI for programming prescriptions.

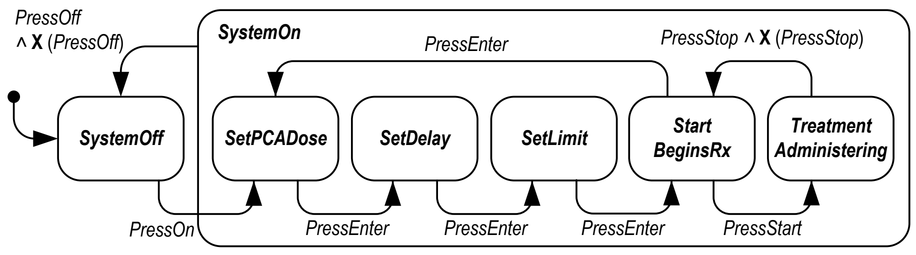
1) Human Mission: The practitioner's mission was to program prescriptions into the pump. The PCA pump prescription was represented by three values: a PCA dosage (*PrescribedPCA*), a minimum delay between dosages (*PrescribedDelay*), and a limit on the total dosage delivered in an hour (*Prescribed1HourLimit*). To control the statespace 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 represented as *Correct* since these were the values operators were attempting to program into the pump.

2) HDI: The HDI for the device represented the mode of the LCD (*InterfaceState*) 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 could be administered (*TreatmentAdministering*). 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*.

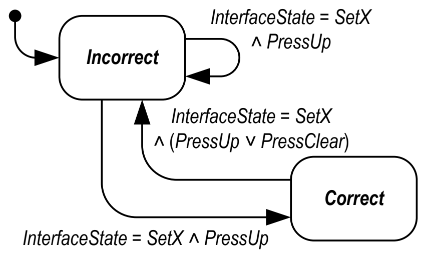
3) Device Automation: The model of the device automation controlled the interface states (Fig. 3a) and displayed values (Fig. 3b) based on internal variables and human actions.

4) HumanTaskBehavior: An instantiated EOFM was created encompassing the following high-level goal directed behaviors for normatively performing activities with the pump (Fig. 4): (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 dosage, delay, and one hour limit).

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. 4e. 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.



(a) *InterfaceState*



(b) *ValueX*

Fig. 3. State transition representation of the formal device automation model. (a) Automation for controlling the interface state. (b) Automation for controlling the displayed value *X* where *X* could represent the PCA dosage, delay, or one hour limit.

The EOFM instance (visualized in Fig. 4) was translated twice into SAL code and incorporated into the larger formal system model: once for normative behavior and once for erroneous human behavior with a maximum of one erroneous transition. The normative behavior model's EOFM representation was 147 lines of code. Its corresponding formal, normative representation was 478 lines of SAL code. The erroneous behavior model was 704 lines of SAL code.

B. Specification and Verification

We use linear temporal logic [5] to specify that, when treatment is administering, the entered prescription should always match that specified by the mission.

$$\mathbf{G}\left(\begin{array}{l}(\mathit{InterfaceMessage} = \mathit{TreatmentAdministering})\\ \Rightarrow \left(\begin{array}{l}\mathit{PCADose} = \mathit{PrescribedPCADose}\\ \wedge\ \mathit{Delay} = \mathit{PrescribedDelay}\\ \wedge\ \mathit{1HourLimit} = \mathit{Prescribed1HourLimit}\end{array}\right)\end{array}\right) \quad (1)$$

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 24 seconds illustrating the following failure sequence:

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 operator erroneously omitted the *ChangeDelayValue* activity, 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 starts treatment by pressing the start button, causing treatment to administer (*TreatmentAdministering*). Thus, the specification has been violated with the pump administering treatment with an unprescribed delay value.

C. Redesign

We can use our method to investigate potential design changes that correct the discovered problem. One possibility is to force the practitioner to review the entered prescription every time it changes. This can be accomplished by having the pump keep track of whether or not the practitioner has reviewed the entered prescription or not and only letting him start the administration of treatment after an entered or changed prescription has been reviewed. This change was made to the PCA pump model, and the resulting model was verified against (1) with the erroneous human behavior model from above. This time the specification verified to true in 22 minutes and 31 seconds having visited 46,706,997 states.

V. Discussion

The erroneous behavior generation process presented here provides a novel means of allowing analysts to automatically determine when potentially unexpected erroneous behavior can result in a violation of a system's safety properties. 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 replicating the observable manifestation of attentional failures associated with Reason's [7] slips: omission, repetition, or commission. 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.

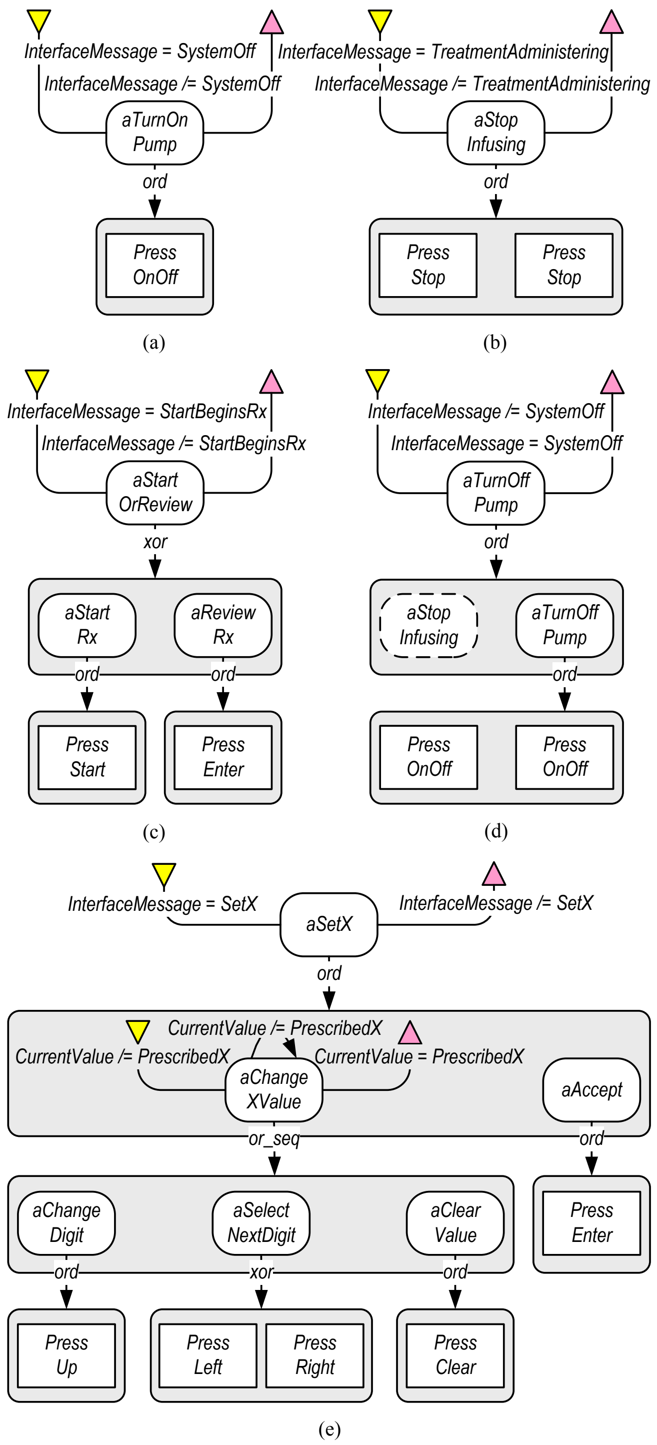


Fig. 4. Visualization of the instantiated EOFM for normatively programming prescriptions into the PCA pump. Activities begin with the letter ”a” and atomic actions do not. Note that (e) represents a generic pattern for programming value *X* into the pump where *X* can be *PCA*, *Delay*, or *1HourLimit*. Also note that the dotted line around *aStopInfusing* indicates that *aTurnOffPump* is referencing the activity *aStopInfusing* defined above it.

This new method has an advantage over the work of Paternò and Santoro [14] in that it allows analysts to automatically consider the impact of erroneous behaviors they may not have anticipated. It also allows for the replication of erroneous behavior that constitute two or more zero-order phenotypes of erroneous action [2] without needing to consider more complex combinations, one of the shortcomings of the automatic method utilized by Bolton and Bass [23]. However, the two generation techniques produce different erroneous behaviors. The techniques discussed here can generate higher level erroneous behaviors based on the incorrect execution of activities. The method in Bolton and Bass [23] can be used to generate lower level erroneous acts, and can allow for the modeling of many more extraneous behaviors as it is capable of generating erroneous actions that are not associated with the currently executing activity. Because of this, analysts may wish to evaluate a system using both techniques, either separately or together, in order to obtain a more complete system evaluation.

A. *Scalability*

The significant increase in the verification times and statespace sizes between the model containing the normative behavior (2 min. 46 sec. and 4,072,083 states) and the one containing the erroneous behavior (22 min. 31 sec. and 46,706,997 states) suggests that the erroneous behavior generation method presented here does not scale well. Thus the method may not be appropriate for the analysis of much larger systems. Future work should investigate how our method scales.

Given that the EOFM to SAL translation process includes all of the intermediary transitions associated with the execution state of activities [11], it is conceivable that the execution state of each activity could be represented exclusively as an expression of the execution state of its 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 [19] add to the statespace. More efficient means of achieving the desired behavior, possibly with different model composition operators, may exist. Future work should investigate these and other methods for potentially improving the scalability of the presented method.

B. *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, it is not capable of replicating ordering errors (a type of commission) for activities contained in an *ord* decomposition. Future work should investigate how to accomplish this.

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.

C. Comparison with Other Methods

Other researchers [27]–[30] have used cognitive modeling to produce erroneous human behaviors in formal models containing HAI. These approaches can evaluate erroneous behaviors related to the repetition of actions, the omission of actions, the commission of correct actions in the wrong context, the replacement of one action with another, the performance of one or more actions out of order, and the performance of unrelated actions; all of which can occur for a variety cognitive reasons. Future work should investigate what the tradeoffs are between this cognitive modeling approach and the task model approaches discussed here and in [23].

Acknowledgment

The majority of the work documented in this manuscript was performed while the first author was pursuing his Ph.D. in systems engineering from the University of Virginia. The project described was supported in part by Grant Number T15LM009462 from the National Library of Medicine (NLM) and NASA Cooperative Agreement NCC1002043. 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] E. Hollnagel, "The phenotype of erroneous actions," *International Journal of Man-Machine Studies*, vol. 39, no. 1, pp. 1–32, 1993.

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

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

[5] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. Leeuwen, A. R. Meyer, M. N., M. Paterson, and D. Perrin, Eds. Cambridge: MIT Press, 1990, ch. 16, pp. 995–1072.

[6] P. M. Jones, "Human error and its amelioration," in *Handbook of Systems Engineering and Management*. Malden: Wiley, 1997, pp. 687–702.

[7] J. Reason, *Human Error*. New York: Cambridge University Press, 1990.

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

[9] F. Paternò, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models," in *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*. London: Chapman and Hall, Ltd., 1997, pp. 362–369.

[10] C. M. Mitchell and R. A. Miller, "A discrete control model of operator function: A methodology for information dislay design," *IEEE Transactions on Systems Man Cybernetics Part A: Systems and Humans*, vol. 16, no. 3, pp. 343–357, 1986.

[11] 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*. doi: 10.1109/TSMCA.2011.2109709, 2011.

[12] R. E. Fields, "Analysis of erroneous actions in the design of critical systems," Ph.D. dissertation, University of York, York, 2001.

[13] 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.

[14] 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.

[15] R. Bastide and S. Basnyat, "Error patterns: Systematic investigation of deviations in task models," in *5th International Workshop on Task Models and Diagrams for Users Interface Design*. Berlin: Springer, 2007, pp. 109–121.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] ——, "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: Human Factors and Ergonomics Society, 2009, pp. 764–768.

[21] 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 Design, Specification, and Verification of Interactive Systems*. Vienna: Springer, 1998, pp. 71–86.

[22] P. A. 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.

[23] M. L. Bolton and E. J. Bass, "Using task analytic models and phenotypes of erroneous human behavior to discover system failures using model checking," in *Proceedings of the 54th Annual Meeting of the Human Factors and Ergonomics Society*. Santa Monica: Human Factors and Ergonomics Society, 2010, pp. 992–996.

[24] 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.

[25] 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.

[26] M. L. Bolton and E. J. Bass, "Using task analytic models to visualize model checker counterexamples," in *Proceedings of the International Conference on Systems Man and Cybernetics*. Piscataway: IEEE, 2010, pp. 2069–2074.

[27] 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.

[28] ——, "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.

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

[30] 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.