Available online at www.sciencedirect.com

SciVerse ScienceDirect

Int. J. Human-Computer Studies 70 (2012) 888–906

International Journal of Human-Computer Studies

www.elsevier.com/locate/ijhcs

Generating phenotypical erroneous human behavior to evaluate human–automation interaction using model checking

Matthew L. Bolton[a,*], Ellen J. Bass[b], Radu I. Siminiceanu[c]

[a]San José State University Research Foundation, NASA Ames Research Center, Moffett Field, CA, USA
[b]Department of Systems and Information Engineering, University of Virginia, Charlottesville, VA, USA
[c]National Institute of Aerospace, Hampton, VA, USA

Received 3 July 2011; received in revised form 16 April 2012; accepted 30 May 2012
Communicated by E. Motta
Available online 9 June 2012

Abstract

Breakdowns in complex systems often occur as a result of system elements interacting in unanticipated ways. In systems with human operators, human–automation interaction associated with both normative and erroneous human behavior can contribute to such failures. Model-driven design and analysis techniques provide engineers with formal methods tools and techniques capable of evaluating how human behavior can contribute to system failures. This paper presents a novel method for automatically generating task analytic models encompassing both normative and erroneous human behavior from normative task models. The generated erroneous behavior is capable of replicating Hollnagel's zero-order phenotypes of erroneous action for omissions, jumps, repetitions, and intrusions. Multiple phenotypical acts can occur in sequence, thus allowing for the generation of higher order phenotypes. The task behavior model pattern capable of generating erroneous behavior can be integrated into a formal system model so that system 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 model) will or will not satisfy safety properties with both normative and generated erroneous human behavior. We present benchmarks related to the size of the statespace and verification time of models to show how the erroneous human behavior generation process scales. We demonstrate the method with a case study: the operation of a radiation therapy machine. A potential problem resulting from a generated erroneous human action is discovered. A design intervention is presented which prevents this problem from occurring. We discuss how our method could be used to evaluate larger applications and recommend future paths of development.

Keywords: Human–automation interaction; Model checking; Task analysis; Human error; Formal methods

1. Introduction

Complex systems depend on goal-oriented human operators interacting with automation in a dynamic environment. Thus, failures in such systems will often result from poor human–automation interaction (HAI). Erroneous human behavior (Hollnagel, 1993), where the human operator does not follow the normative procedure for interacting with a system has been associated with failures in aviation (BASI, 1998; FAA Human Factors Team, 1996; Hughes and Dornheim, 1995; Johnson and Holloway, 2004), process control (O'Hara et al., 2008), medicine (Kohn et al., 2000), and other safety critical domains. A number of model-driven analysis techniques have been developed to give engineers methods and tools for evaluating HAI in system design and analysis. However, there are gaps in the techniques that attempt to use formal methods to evaluate the impact of human behavior on system safety. We describe these model-driven approaches before presenting a new method that attempts to address some of their shortcomings.

1.1. Model-driven design and analysis

Model-driven design and analysis techniques (Hussmann et al., 2011) use models of the automation, human–device

*Corresponding author. Tel.: +1 650 604 2505; fax: +1 650 604 3729.
E-mail address: mlb4b@virginia.edu (M.L. Bolton).

1071-5819/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.ijhcs.2012.05.010

interfaces, 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 specification, modeling, and verification of systems (Wing, 1990). While there are a number of different ways in which models can be both manually and automatically verified, two particular computer software technologies, automated theorem provers and model checkers, have been shown to be useful for the formal verification of large complex systems. Theorem proving is a deductive technique that closely resembles the traditional pencil-and-paper proof activity: from a set of axioms, using a set of inference rules, one builds theories and proves theorems to verify correctness claims about the system under investigation with the help of a proof assistant program. Model checking is an automated approach used to verify that a formal model of a system satisfies a set of desired properties (a specification) (Clarke et al., 1999). A formal model describes a system as a set of variables and transitions between variable values (states). Specification properties are commonly represented in a temporal logic, usually Linear Temporal Logic or Computation Tree Logic, using the variables that describe the formal system model to construct propositions. Verification is the process of proving that the system meets the properties in the specification. Model checking performs this process automatically by exhaustively searching a system's statespace to determine if these criteria hold. If there is a violation, an execution trace is produced (a counterexample). This counterexample 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 the violation.

Formal, model-based approaches have been used to model human–device and human–computer interfaces both with and without the underlying automation (Abowd et al., 1995; Degani, 2004; Degani et al., 2011; Degani and Heymann, 2002; Dix et al., 2008; Dwyer et al., 1997, 2004; Harel, 1987; Harrison and Duke, 1995; Parnas, 1969; Sherry and Ward, 1995; Thimbleby and Gow, 2007a). These models can then be used in a variety of formal analyses to evaluate the HAI. For example: researchers have formulated a number of generic temporal logic properties for evaluating interface usability with a model checker (Abowd et al., 1995; Berstel et al., 2005; Campos and Harrison, 1997, 2009; Paternò, 1997); a variety of techniques have been developed for detecting potential mode confusion[1] in automated systems (Bredereke and Lankenau, 2005; Buth, 2004; Campos and Harrison, 2001, 2011; Degani and Heymann, 2002; Javaux, 2002; Joshi et al., 2003; Leveson et al., 1997; Rushby, 2002; Wheeler, 2007); graph theory can be used to evaluate usability (Thimbleby and Gow, 2007b); methods have been developed for generating test cases that will guarantee coverage of the devices features and displays (Duan et al., 2010; Giannakopoulou et al., 2011); code for implementing modeled human–computer interfaces can be automatically generated (Berstel et al., 2005); and human–device interfaces can be automatically generated to ensure certain usability properties are maintained (Combéfis et al., 2011; Gimblett and Thimbleby, 2010; Heymann and Degani, 2007).

[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 (Norman, 1990; Sarter and Woods, 1995).

1.2. Formally evaluating the impact of human behavior on system safety

Of particular relevance to this work are model-driven analysis techniques that attempt to use formal methods and human-behavior modeling to evaluate system safety. In these approaches, the human operator's behavior is determined by device interface models, cognitive models of human behavior, or task-analytic models of human behavior.

1.2.1. Using device interfaces to define human behavior

In the simplest approach, the human operator's behavior is determined by the human–device interface such that any input acceptable by the interface is allowed. The human–device interface and/or automation behavior, including its response to human behavior, is modeled formally. Formal verification is then used to determine if there is any way the system could ever violate a safety property (Abowd et al., 1995; Campos and Harrison, 1997, 2008; Thomas, 1994).

This approach is powerful in that it will tell an analyst whether or not there is ever a situation where human behavior could contribute to a system failure. However, it gives the analyst little insight into what the human operator was doing when a failure occurred. Additionally, this approach can find failure sequences that a human operator may never actually perform. Finally, there are many safety critical systems where the human operator will always be able to cause a failure. For example, an aircraft pilot is always capable of causing a crash.

1.2.2. Using cognitively driven models of human behavior

Another approach explicitly includes human behavior as part of the larger formal system model. In the work of Blandford et al. (Blandford et al., 2004, 1997; Butterworth et al., 1998, 2000) human behavior is driven by a formal model of human cognition (based on Programmable User Models; Young et al., 1989). These cognitive models encompass the human operator's goals, his or her beliefs and knowledge about the operation of the system, the information available to him or her from the human–device interface, and the actions he or she can use to interact with the system. When such models are integrated into a formal system model, system safety and liveness properties can be evaluated using model checkers and/or automated theorem provers. With this approach,

erroneous behavior can result from the human cognitive process. Cognitive models can be used to generate simple erroneous behaviors such as the repetition, omission, intrusion, replacement, mis-sequencing or mis-timing of actions (Curzon and Blandford, 2002). They can also be used to generate post completion errors (Curzon and Blandford, 2004), a type of omission which occurs when a human operator successfully completes his or her primary goal with a device but fails to perform supplemental actions to fully complete the task (Byrne and Bovair, 1997). These types of analyses can also be used to determine how different types of human operators (novice or expert) may impact system performance based on how their knowledge of, beliefs about, and goals for the operation of the system can result in repetitions of actions or post completion errors (Curzon et al., 2007). The architecture has been extended to a generic cognitive framework that models the effect of salience, cognitive load, and the interpretation of spatial cues and assesses whether they result in erroneous human behavior (Rukšėnas et al., 2008; Rukšenas et al., 2009; Rukšėnas et al., 2007). Habituation, impatience, and carefulness have also been incorporated into these types of models (Basuki et al., 2009). Means of evaluating distributed cognition formally have also been explored (Masci et al., 2011).

Because they explicitly model human behavior and because they represent the cognitive basis for erroneous behavior, the techniques discussed here give analysts additional insights into why failures occur. However, these analyses require each cognitive mechanism to be incorporated into the model. As such, they miss un-modeled behaviors. Finally, the cognitive models employed in these techniques are less commonly used than other human behavior modeling techniques such as task models.

1.2.3. Using models of task analytic human behavior

A third approach for assessing the impact of human behavior on system safety incorporates task analytic models into formal verification analyses. When designing the procedures, displays, controls, and training associated with the HAI of an automated system, human factors engineers typically use task analytic methods to describe the normative human behaviors required to control the system (Kirwan and Ainsworth, 1992). Information from these methods can be used to model the mental and physical activities that operators use to achieve goals with the system.

Task analytic models are computational structures represented as a hierarchy of activities that decompose into other activities and (at the lowest level) atomic actions. Task analytic models such as ConcurTaskTrees (CTT) (Paternò et al., 1997), Operator Function Model (OFM) (Mitchell and Miller, 1986), Enhanced Operator Function Model (EOFM) (Bolton et al., 2011), User Action Notation (UAN) (Hartson et al., 1990), AMBOSS (Giese et al., 2008), and HAMSTERS (Martinie et al., 2011) can be represented using discrete graph structures. In these models, strategic knowledge (condition logic) controls when activities can execute, and modifiers between activities or actions control how they execute in relation to each other.

Because they can be represented discretely, task analytic models can be used to include human behavior in formal system models. A number of researchers have incorporated task analytic models into formal system models of human–automation interactive systems by either modeling task behavior natively in the formal notation (Basnyat et al., 2007, 2008; Campos, 2003; Gunter et al., 2009) or translating task analytic models implemented in task analytic representations (such as CTT, EOFM, or UAN) into the formal notation (Aït-Ameur and Baron, 2006; Aït-Ameur et al., 2003; Bolton and Bass, 2009b, 2010a; Bolton et al., 2011; Fields, 2001; Palanque et al., 1996; Paternò and Santoro, 2001; Paternò et al., 1998). This allows system safety properties to be verified in light of the modeled, normative human behavior, thus giving researchers a means of proving that (assuming the representativeness of their system model) the system will always operate safely if the human operators adhere to the modeled behavior.

These analyses have the advantage that they allow engineers to reuse the information gleaned from task analytic methods. However, these approaches are limited because the models only encompass normative human behavior and thus the analyses provide no insight into the impact of erroneous human behavior. To address this, researchers have investigated how erroneous human behavior can be systematically incorporated into task analytic behavior models, often with the explicit or implied ability to evaluate the impact of the generated erroneous human behavior on system properties using formal verification. For example, several researchers have demonstrated how task analytic models of normative human behavior can be manually permuted to include erroneous human behaviors: either simple deviations from the performance of actions in a task (Bolton et al., 2008; Fields, 2001) or more complicated, cognitively significant divergences such as post completion errors (Basnyat and Palanque, 2005; Palanque and Basnyat, 2004; Paternò and Santoro, 2002). Bastide and Basnyat (2007) and Fields (2001) have taken this a step further by introducing "error patterns" representing erroneous behavior deviations from normative task behavior models, where the manual application of the transformation rules will automatically incorporate the erroneous behavior in the desired location. This approach allows an expert on erroneous human behavior to examine a normative human task behavior model and selectively include erroneous human behaviors in locations where he or she thinks they might occur. The impact of the included erroneous behaviors can then be evaluated with formal verification (Bolton et al., 2008; Fields, 2001).

The approaches covered in this section have an advantage over the cognitive modeling-based techniques discussed above in that they leverage information commonly

developed in the task analysis process. Further, the erroneous behavior manually incorporated into the task behavior models can be included for both cognitive or non-cognitive reasons. However, the fact that erroneous behaviors must be manually included in these analyses means that these methods will not provide insight into the impact of unanticipated erroneous human behavior, while the more advanced cognitive models can.

1.3. Objectives

Clearly, analysts have some options if they want to evaluate the impact of human behavior of system safety formally. What is missing is an approach that would allow engineers to reuse task analytic behavior models to evaluate the impact of potentially unanticipated, erroneous deviations from normative behavior. Such an approach would avoid the pitfalls of the human device interface and cognitive modeling approaches. Further, it would be able to leverage the advantages of using task analytic behavior models without forcing the analyst to select which erroneous behaviors to include.

In this work, we present a technique capable of performing such an analysis. Our technique uses a well established taxonomy of erroneous human behavior (based on Hollnagel, 1993) to automatically generate erroneous behavior within instantiated EOFM task analytic models of normative human behavior. Because instantiated EOFMs can be automatically translated into formal modeling notation (Bolton et al., 2011), the impact of the generated, and potentially unanticipated, erroneous human behavior on system safety can be evaluated using formal verification with a model checker. In the remainder of this paper we discuss our analysis method for the formal verification of HAI and show how erroneous human behavior can be automatically generated and evaluated as part of this process. We report benchmarks which show how our method scales with respect to the number of available human actions and the maximum number of erroneous acts allowed with different task structures. We demonstrate how our method can be used to find violations of system safety properties due to automatically generated erroneous human behavior with a radiation therapy machine example. We show how our method can be used to explore design interventions which eliminate the discovered problem. Finally, we discuss our results and recommend avenues of future research.

2. Using EOFM in the formal verification of HAI

We have developed a method (Bolton and Bass, 2009b) 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 (the activities and actions the human operator uses to achieve mission goals), human–device interfaces (displays and controls available to the human operator through the device), device automation (underlying device behavior), and the operational environment (Bolton and Bass, 2010a). In our analysis framework, human task models are created using an intermediary language called

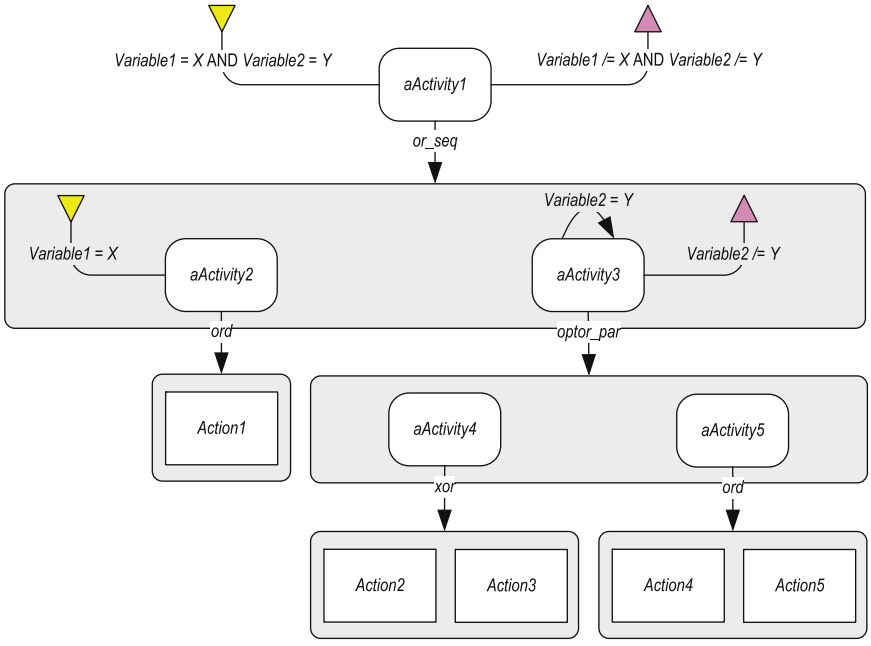

Fig. 1. An example of the visual representation of a task structure in an instantiation of an EOFM. Activity *aActivity1* has both a precondition and a completion condition. It decomposes into activities *aActivity2* and *aActivity3* with an *or_seq* decomposition operator. *aActivity2* has a precondition and decomposes into *Action1* with an *ord* decomposition operator. *aActivity3* has both repeat and completion conditions. It decomposes into *aActivity4* and *aActivity5* with an *optor_par* operator. *aActivity4* and *aActivity5* each decompose into two actions, *Action2* and *Action3* with an *xor* decomposition for the former and *Action4* and *Action5* with an *ord* decomposition for the latter.

Enhanced Operator Function Model (EOFM) (Bolton et al., 2011), an XML-based human task modeling representation derived from the Operator Function Model (OFM) (Mitchell and Miller, 1986; Thurman et al., 1998). EOFMs are hierarchical representations of goal-driven activities that decompose into lower level activities, and finally, atomic actions. EOFMs express task knowledge by explicitly specifying the conditions under which human operator activities can be undertaken: what must be true before they can execute (preconditions), when they can repeat (repeat conditions), and when they have completed (completion conditions). Any activity can decompose into one or more other activities or one or more actions. 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 can be represented visually as tree-like graphs (see Fig. 1). 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 (Bolton et al., 2011) are used:

- *ord* — all activities or actions in the decomposition must execute in the order they appear;
- *or_seq* — one or more of the activities or actions in the decomposition must execute, where only one activity or action can be executing at a given time;
- *optor_par* — zero or more of the activities or actions in the decomposition must execute, where the execution of activities or actions can overlap; and
- *xor* — exactly one activity or action in the decomposition must execute.

Conditions 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 Bolton et al. (2011).

EOFM has formal semantics which specify how an instantiated EOFM model executes (Bolton et al., 2011). Specifically, each activity or action can have 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; the state of its immediate parent, its siblings (activities or actions contained in the same decomposition), and its immediate children in the hierarchy; and the decomposition operators that connect the activity to its parent and its children.

Instantiated EOFM task models can be automatically translated (Bolton et al., 2011) into the language of the Symbolic Analysis Laboratory (SAL) (De Moura et al., 2003) using the language's formal semantics where they can be integrated into a larger system model using a defined architecture and coordination protocol (Bolton and Bass, 2010a; Bolton et al., 2011). 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 Bolton and Bass, 2010b). These techniques have been successfully used to evaluate a patient controlled analgesia pump (Bolton and Bass, 2009b, 2010a), an automobile cruise control system (Bolton and Bass, 2010b; Bolton et al., 2011), and an aircraft on approach (Bolton and Bass, in press).

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

3. Erroneous behavior generation

3.1. Phenotypes of erroneous human behavior

Erroneous human behaviors have been classified based on how they arise from human decision making, cognition, and task execution (Hollnagel, 1993; Jones, 1997; Norman, 1988; Reason, 1990). Task analytic behavior models generally do not model low level perceptual, motor, and cognitive processes (exceptions are discussed by John and Kieras, 1996). However, they do model human task behavior hierarchically down to the atomic, observable action level. For this reason, Hollnagel's (Hollnagel, 1993) phenotypes of erroneous action are appropriate.

Hollnagel (1993) showed that erroneous human behaviors can be classified based on how they observably manifest as a divergence from a plan or normative sequence of actions (a task). He called these the phenotypes of erroneous human action. In this taxonomy, erroneous behaviors are composed of one or more erroneous acts, all capable of being detected by observing the performance of a single act in a plan. These "zero-order" phenotypes include: 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). These serve as the building blocks for additional, "first-order", phenotypes: spurious intrusions (actions are added to plans via zero-order intrusions), jumping forward or backward in a plan (multiple skips or jumps), place losing (performing planned actions in an arbitrary order via multiple skips and jumps), recovery (performing previously omitted actions via multiple jumps), capture (performing part of another action sequence in the wrong place via multiple intrusions), reversal (reversing the execution of two adjacent actions via a skip and a jump backward), and time compression (multiple premature starts and/or premature finishes).

4. Phenotypical erroneous human behavior generation

Hollnagel's phenotypes (Hollnagel, 1993) for delays and premature starts and finishes refer to time explicitly, which is not currently supported by our method (Bolton and Bass, 2009b). However, all of the other zero-order phenotypes relate to the performance or non-performance of actions; all of which are compatible with the formal semantics and structure of the EOFM. Thus the intent of this work was to develop an erroneous behavior generation process capable of replicating Hollnagel's (Hollnagel, 1993) zero-order phenotypes of erroneous human behavior for omitting, skipping, re-performing, repeating, or intruding an action for each original action in an instantiated EOFM. To allow for more complex erroneous human behaviors, erroneous behavior generation must be capable of chaining zero-order erroneous human acts. An unconstrained number of erroneous acts could result in an unbounded human task behavior model, which would eliminate the benefit of including human behavior in the model. Thus, the erroneous behavior generation process must support a mechanism for constraining the number of erroneous acts that can be performed in the formally translated erroneous human behavior model. To facilitate analysis with our method, the erroneous behavior generation structure should be represented in the EOFM language thus making it compatible with the EOFM to SAL translation process and counterexample visualization.

To accomplish these goals, we replace each action in an instantiated EOFM task model with an EOFM task structure that generates erroneous behavior. Thus, for every given action in the original normative structure, the new structure can allow for the original action and/or one or more zero-order phenotypes. The ability to perform multiple erroneous acts allows zero-order erroneous acts to be chained together to form higher order erroneous behaviors. A counter keeps track of the number of erroneous acts, and the number of erroneous acts is limited by a maximum. What follows is a discussion of the details for implementing erroneous behavior generation in this manner, and how it was adapted into our method for the formal verification of HAI.

To generate erroneous actions, zero-order phenotypes for omissions, skips, re-performances, repetitions, and intrusions are incorporated into an instantiated EOFM by replacing each atomic action ($Action^{x}$) with a customized structure ($Action^{x'}$) (Fig. 2). This design includes an upper bound on the number of erroneous acts ($EMax$) and a variable ($ECount$) that keeps track of the number of erroneous acts that the task model has performed. Any activity that represents an erroneous act has a precondition asserting that it cannot be performed unless the current number of performed erroneous actions is less than the maximum ($ECount < EMax$). Every time an activity representing an erroneous act executes, $ECount$ is incremented by one ($ECount$++).

$Action^{x'}$ decomposes into several additional activities, allowing $Action^{x'}$ to complete execution if one or more of these activities executes (via the *or_seq* decomposition operator). *CorrectAction* allows the original correct action

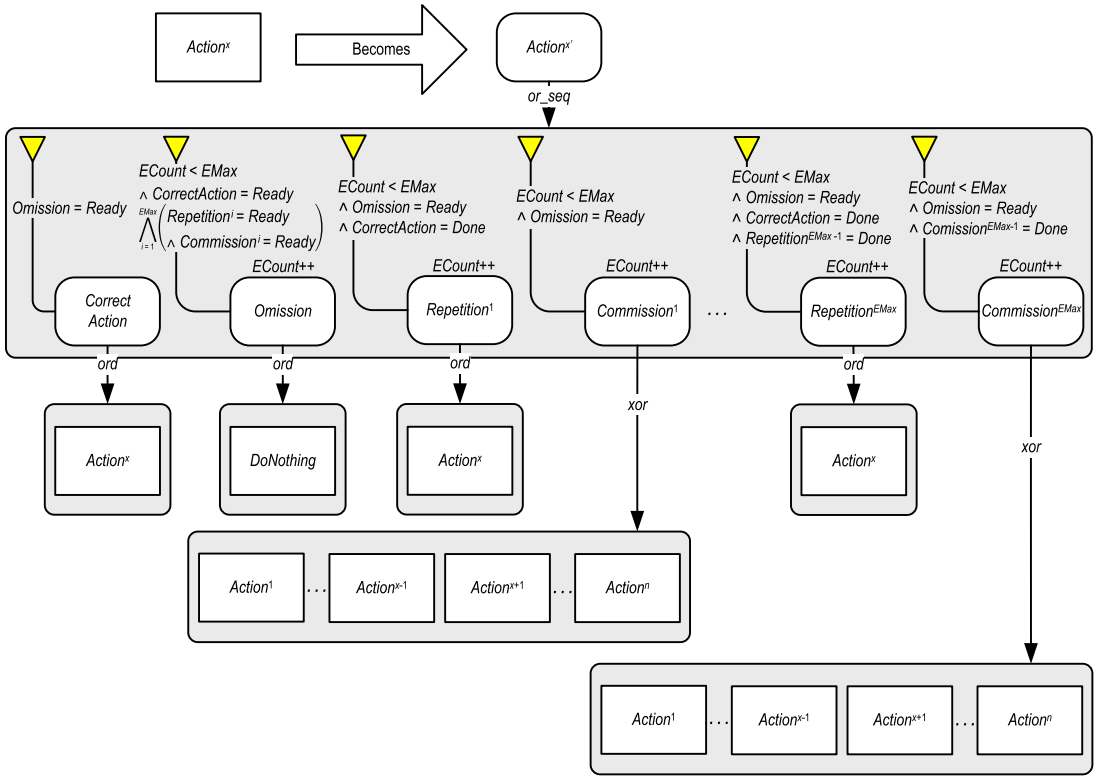

Fig. 2. Visualization of the EOFM structure used to generate zero-order phenotypical erroneous human behavior.

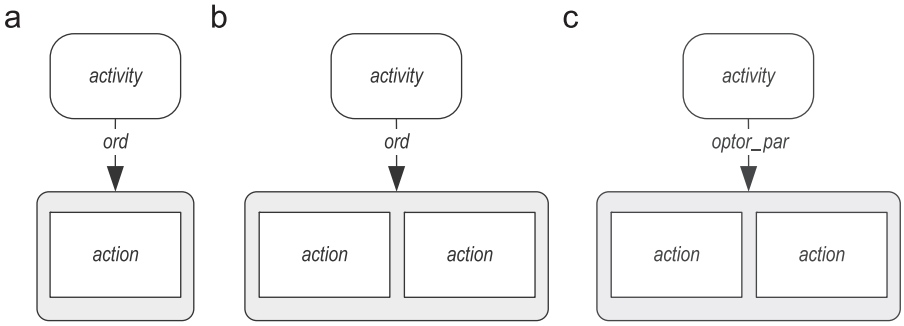

Fig. 3. Instantiated EOFM normative task behavior structures used as inputs to verification benchmark experiments: (a) A single activity decomposing into a single action with an *ord* decomposition; (b) A single activity decomposing into two actions with an *ord* decomposition; and (c) A single activity decomposing into two actions with an *optor_par* decomposition.

($Action^x$ to be performed). The *Omission* activity allows a human operator to perform the erroneous act of omitting the original action, represented as the *DoNothing* action. The *Repetition* activities each allow the correct action to be performed erroneously multiple times, one for each activity. The *Commission* activities each allow any other single erroneous action to be performed (via the *xor* decomposition operator) where the set of erroneous actions corresponds to the n human actions available to the human operator. There are *EMax Commission* and *Repetition* activities, thus allowing for up to *EMax* erroneous actions to occur in place of or in addition to the original action.

Aside from keeping the number of erroneous acts from exceeding the maximum, the precondition of each activity in this structure ensures that there is only one way to produce a given sequence of actions for each instance of the structure. An omission can only execute if no other activity in the structure has executed (all other activities must be *Ready*) and every other activity can only execute if there has not been an omission ($Omission = Ready$). The first repetition activity ($Repetition^1$) cannot execute unless the correct action activity is done ($CorrectAction = Done$). Every subsequent repetition activity ($Repetition^i$) cannot execute unless the previous repetition activity has executed ($Repetition^{i-1} = Done$). Similarly, every commission activity $Commission^i$ where $i > 1$ cannot execute unless the previous commission activity is done ($Commission^{i-1} = Done$).

This design allows the specified zero-order erroneous behavior phenotypes to be generated when the EOFM executes. Skips and omissions occur through the execution of the *Omission* activity. Repeating the current action can occur through a *Repetition* activity. Re-performing a previously completed action or performing an intrusion can occur by executing either a previously performed action or some other action (respectively) through one of the *Commission* activities. Multiple erroneous actions can occur before, after, or instead of the correct action due to the *or_seq* decomposition and multiple erroneous acts can occur between erroneous behavior generating structures for different actions. Thus, the use of this structure allows for single, zero-order erroneous actions as well as more complicated (first-order or second-order) erroneous behaviors to be generated.

Our Java-based EOFM to SAL translator (Bolton et al., 2011) was modified to incorporate the erroneous behavior generation structure into any instantiated EOFM. The translator takes a normative human behavior model (an instantiated EOFM XML file) and the maximum number of erroneous acts (*EMax*) as input and traverses the EOFM structure, replacing each action with its corresponding erroneous behavior generative structure (Fig. 2). To accommodate the verification process, the translator represents *EMax* as a constant and the range for the number of possible erroneous acts (0 to *EMax*) as a type. It modifies each human operator by adding a variable representing the current number of performed erroneous acts (*ECount*) and a *DoNothing* human action.

The translator produces two files as output. The first is an EOFM XML file representing the created erroneous human behavior model (separate from the model of normative behavior). The second is the translated SAL representation of this model.

5. Testing and benchmarks

Benchmarking was performed to shed light on how the erroneous behavior generation process impacted model complexity and scalability. For these analyses, we wanted to develop an understanding of how the erroneous behavior generation process contributed to the statespace size and verification times of task behavior models instantiated in EOFM. The complexity added to an instantiated EOFM task behavior model can be influenced by two factors. The first is the complexity of the erroneous behavior generation structure that replaces each action. This, in turn, can be influenced by the maximum number of erroneous acts (*EMax*), which determines how many *Commission* and *Repetition* activities are in the erroneous behavior generation structure; and the number of available human actions, which populate the generating structure's *Commission* activities. The second factor that impacts complexity is the relationship between each of the generating structures based on the decomposition operators that control it (Bolton et al., 2011).

We conducted a series of verifications using three different instantiated EOFM normative task behavior

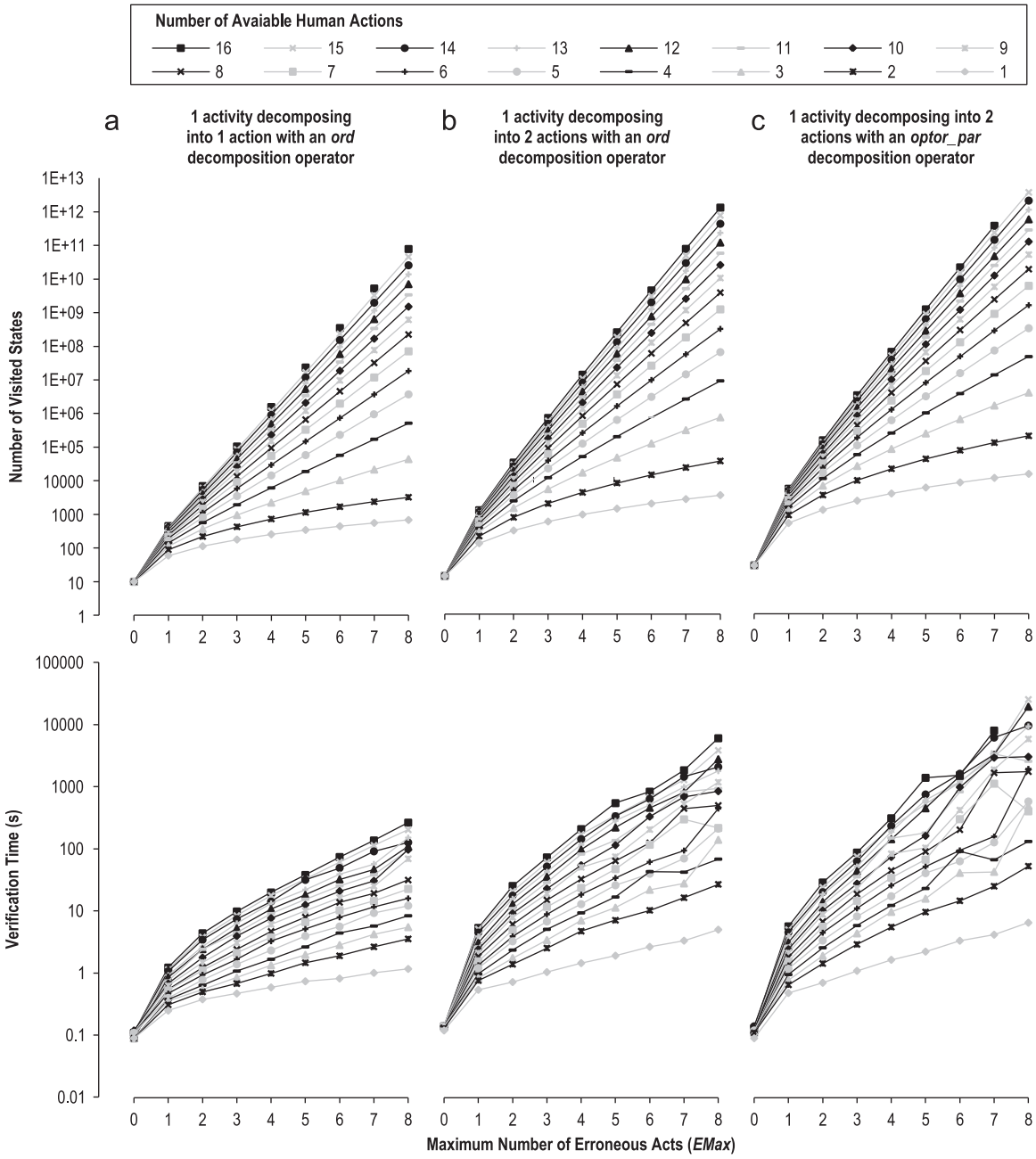

Fig. 4. Plot of the verification results (verification time in seconds and the number of visited states) vs. the maximum number of erroneous acts (*EMax*) for between 1 and 16 available human actions for each of the three task structures ((a), (b), and (c)) from Fig. 3 on log-linear coordinates.

models (Fig. 3). The structure shown in Fig. 3(a) represents the simplest valid instance of an EOFM: where a single activity decomposes into a single action. Thus, benchmarks collected from verifications using this model as input provide insight into the scalability of the erroneous behavior generation structure in isolation. The structures shown in Fig. 3(b) and (c) represent the simplest EOFM task structure in which there is more than one action. The *ord* decomposition operator[2] (used in Fig. 3(b)) has the least impact on model complexity while the *optor_par* decomposition operator[3] (used in Fig. 3(c)) has the most (Bolton et al., 2011). Thus, the model in Fig. 3(b) supports lower bound analysis of the impact the erroneous behavior structures has on verification results while the model in Fig. 3(c) supports an upper bound analysis.

In the analysis, we varied the number of available human actions (between 1 and 16) and the maximum number of erroneous acts (between 0 and 8) for each of the task structures ((a), (b), and (c)) from Fig. 3. We used our translator to create a formal SAL model for each distinct

[2]All of the sub-activities or actions must execute sequentially in the order they are listed.

[3]Zero or more of the sub-activities or actions can execute in any order and the execution of activities or actions can overlap.

combination. Each model was formally verified once against a valid specification using SAL-SMC on a computer workstation with a 3.0 GHz dual-core Intel Xeon processor and 16 GB of RAM running the Ubuntu 9.04 desktop, where both the total number of visited states and the verification time, as reported by SAL-SMC, were recorded. These results are presented in Fig. 4. Note that the results for task structure (c) with an *EMax* of 8 and 16 human actions are not reported because the verification exceeded the memory capacity of the computer.

A visual inspection of these results suggested that, for all three task structures, the statespace and verification times initially increased with the introduction of the erroneous generation structure ($EMax=0$ to $EMax=1$). It then grew exponentially with *EMax*, where the rate of exponential growth was higher for larger numbers of allowable human actions. We confirmed this exponential relationship by calculating the R^2 of the log of both the number of visited states and the verification time as predicted by the value of *EMax* (for $EMax > 0$) for each number of available human actions (Table 1). R^2 ranged from 0.938 to 1. All were significant at $p < 0.05$. Thus, both statespace and verification time increase at an approximately exponential rate with *EMax*.

As expected, models based on task structure (a) (which had only one action in its normative configuration) were less complex (had fewer visited states) for comparable numbers of human actions and *EMax* values than the models based on the other two task structures (which both had two actions in their normative configuration). Similarly, models based on task structure (b) which had the more restrictive *ord* decomposition operator had smaller statespaces for comparable parameters than models based on task structure (c) which utilized *optor_par*. Verification times generally increased with the complexity of the model.

6. Application

To demonstrate how this type of erroneous behavior generation can be used, we evaluate a model of a human operated radiation therapy machine[4] based loosely on the Therac-25, a radiation therapy machine responsible for the deaths of several patients in the 1980s (Leveson and Turner, 1993). This device is a room-sized, computer-controlled, medical linear accelerator. It has two treatment modes: the electron beam mode is used for shallow tissue treatment, and the x-ray mode is used for deeper treatments—requiring electron beam current approximately 100 times greater than that used for the electron beam mode. The x-ray mode uses a beam spreader (not used in electron beam mode) to produce a uniform treatment area and attenuate radiation of the beam. An x-ray beam treatment application without the spreader in place can deliver a lethal dose of radiation.

[4]A full listing of all of the model code used in this process can be found at http://cog.sys.virginia.edu/formalmethods/

In the following, we describe the components of the formal model and the specification that were used as the basis for all analyses. We then describe a sequence of verification analyses in which the maximum number of erroneous acts (*EMax*) was iteratively increased. Verification results and statistics are reported for each verification analysis.

6.1. Formal model

6.1.1. Human–device interface

The human–device interface model (Fig. 5) includes interactions with the five relevant keyboard keys ('X', 'E', Enter, Up and 'B') and the information on the monitor. The interface state (*InterfaceState*; Fig. 5(a)) starts in *Edit* where the human operator can press 'X' or 'E' (*PressX* or *PressE*) to select the x-ray or electron beam mode and transition to the *ConfirmXrayData* or *ConfirmEBeamData* states, respectively. When in the *ConfirmXrayData* or *ConfirmEBeamData* states, the appropriate treatment data are displayed (*DisplayedData*; Fig. 5(b)), and the human operator can confirm the displayed data by pressing enter (advancing to the *PrepareToFireXray* or *PrepareToFireEBeam* states) or return to the *Edit* state by pressing up (*PressUp*) on the keyboard. In the *PrepareToFireXray* or *PrepareToFireEBeam* states, the human operator must wait for the beam to become ready (*BeamState*; Fig. 5(c)), at which point he or she can press 'B' (*PressB*) to administer treatment by firing the beam. This transitions the interface state to *TreatmentAdministered*. The operator can also return to the previous data confirmation state by pressing up.

6.1.2. Device automation

The device automation model (Fig. 6) controls the power level of the beam, the position of the spreader, and the firing of the beam. The power level of the beam (*BeamLevel*; Fig. 6(a)) is initially not set (*NotSet*). When the human operator selects the x-ray or electron beam treatment mode, the power level transitions to the appropriate setting (*XrayLevel* or *EBeamLevel*, respectively). However, if the human operator selects a new power level, there is a delay in the transition to the correct power level, where it remains in an intermediary state (*XtoE* or *EtoX*) at the old power level before automatically transitioning to the new one. The position of the spreader (*Spreader*; Fig. 6(b)) starts either in or out of place (*InPlace* or *OutOfPlace*). When the human operator selects the x-ray or electron beam treatment mode, the spreader transitions to the appropriate setting (*InPlace* or *OutOfPlace* respectively). The firing state of the beam (*BeamFireState*; Fig. 6(c)) is initially waiting to be fired (*Waiting*). When the human operator fires the beam (pressing 'B' when the beam is ready), the beam fires (*Fired*) and returns to its waiting state.

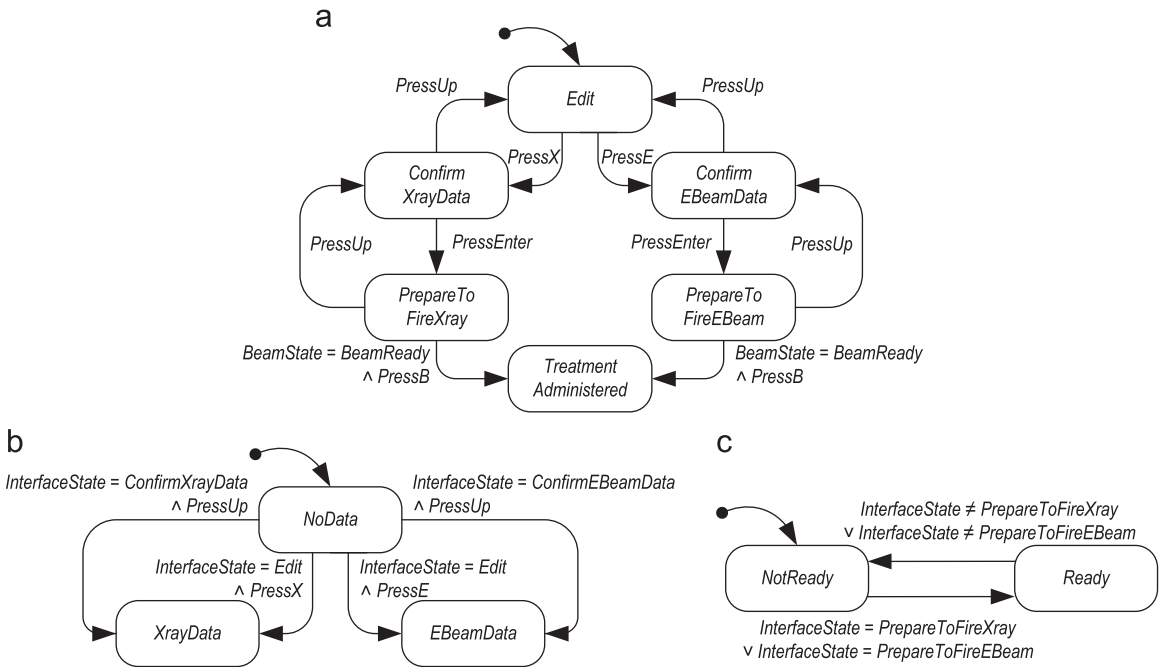

Fig. 5. State transition representation of the formal human-device interface model. Rounded rectangles represent states. Arrows between states represent transitions. Dotted arrows indicate initial states. (a) *InterfaceState*. (b) *DisplayedData*. (c) *BeamState*.

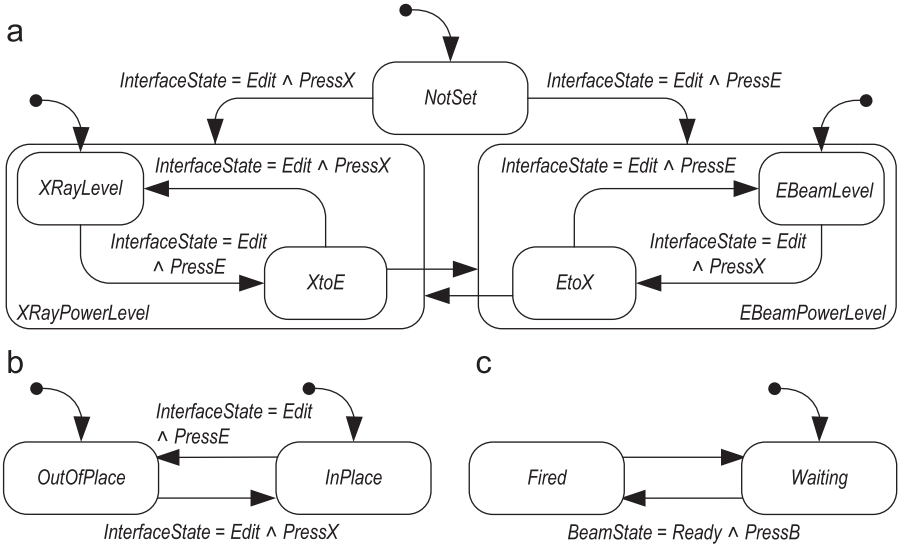

Fig. 6. State transition representation of the formal device automation model. (a) *BeamLevel*. (b) *Spreader*. (c) *BeamFireState*.

6.1.3. Human mission

The human mission identifies the goals the human operator is attempting to achieve when interacting with the device: the type of treatment the human operator is attempting to administer. In the formal model, this is represented as a variable (*TreatmentType*) that can initialize to either *Xray* or *EBeam*.

6.1.4. Normative human task behavior

Three goal-directed task models describe the administration of treatment with the radiation therapy machine (Fig. 7): selecting the treatment mode (*aSelectXorE*; Fig. 7(a)), confirming treatment data (*aConfirm*; Fig. 7(b)), and firing the beam (*aFire*; Fig. 7(c)).

These models access input variables from the human–device interface (the interface state (*InterfaceState*), the displayed treatment data (*DisplayedData*), and the ready status of the beam (*BeamState*)) and the mission (treatment type (*TreatmentType*)) to generate the human actions for pressing the appropriate keyboard keys.

When the interface is in the edit mode (*aSelectXorE*; Fig. 7(a)), the practitioner can select the appropriate treatment mode based on the mission *TreatmentType* by performing either the *PressX* or *PressE* actions. When the

Fig. 7. Visualization of the EOFMs for interacting with the radiation therapy machine: (a) selecting the treatment mode (*aSelectXorE*), (b) confirming treatment data (*aConfirm*), and (c) firing the beam (*aFire*).

interface is in either of the two data confirmation states (*aConfirm*; Fig. 7(b)) the practitioner can choose to confirm the displayed data (if the displayed data correspond to the desired treatment mode) by pressing enter or return to the *Edit* state by pressing up. When the interface is in either of the states for preparing to fire the beam (*aFire*; Fig. 7(c)), the practitioner can choose to fire the beam if the beam is ready by pressing 'B' or return to the previous interface state by pressing up.

6.2. Specification

We specify that we never want the radiation therapy machine to irradiate a patient by administering an unshielded x-ray treatment using Linear Temporal Logic as shown in (1).

$$\mathbf{G}\neg\left(\begin{array}{l} BeamFireState = Fired \\ \wedge\, BeamLevel = XRayPowerLevel \\ \wedge\, Spreader = OutOfPlace \end{array}\right) \quad (1)$$

This can be interpreted as "we never want the beam to fire when it is at the x-ray level and the spreader is out of place."

6.3. Analyzing system safety with normative human behavior

6.3.1. Translation

The EOFM instance was translated into SAL code using our modified translator and incorporated into the larger

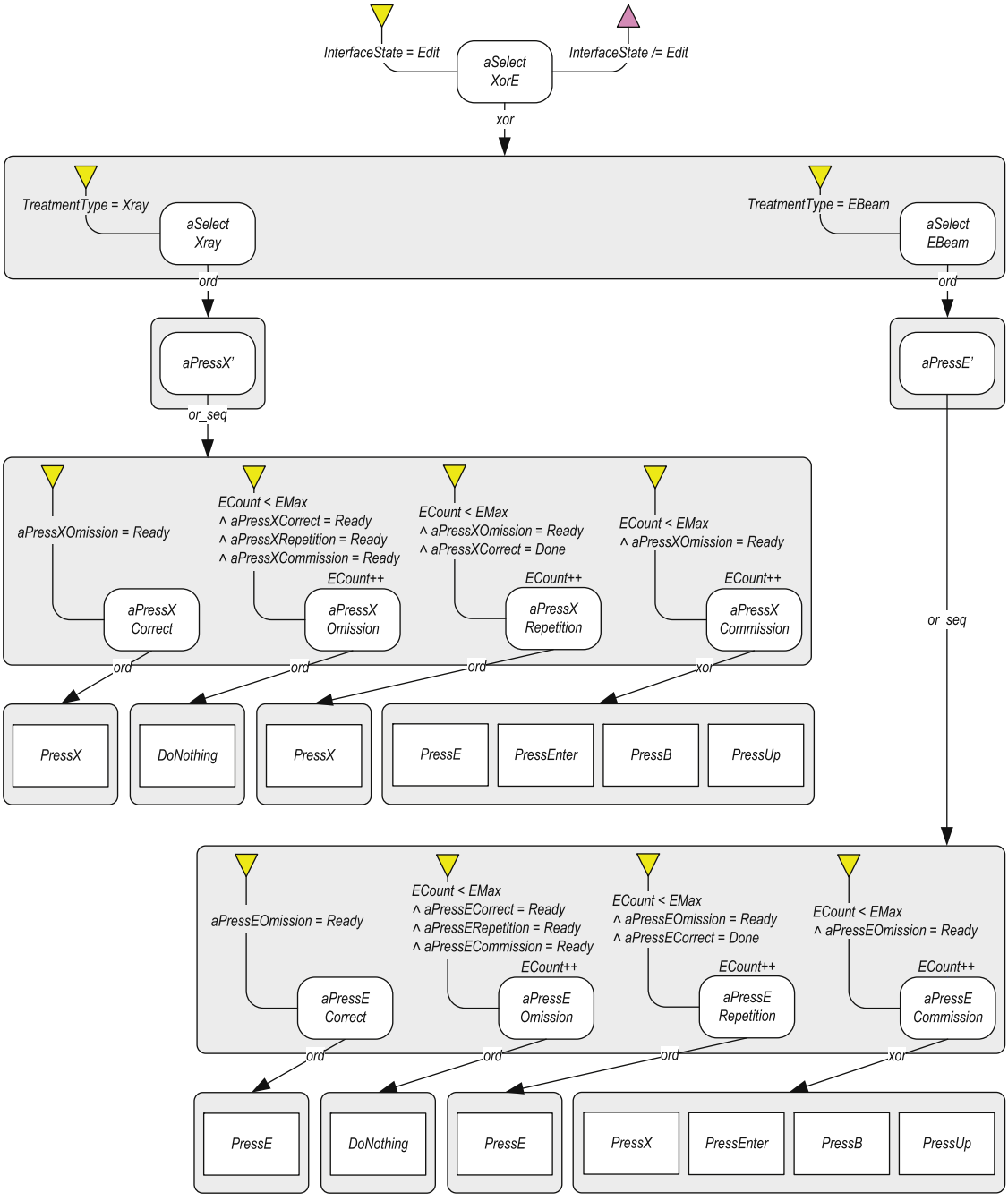

Fig. 8. Visualization of the EOFM structure that was produced when the erroneous behavior generating structure with a maximum of one erroneous act ($EMax=1$) was incorporated into the *aSelectXorE* activity (Fig. 7(a)).

formal system model using only the normative human behavior ($EMax=0$). The normative behavior model's EOFM representation was 74 lines of XML code. Its corresponding formal representation was 166 lines of SAL code. The majority of the increase in code size is due to the fact that the SAL model must explicitly represent the formal semantics of the EOFM and the coordination protocol that allows the task model to interact with other system elements.

6.3.2. Verification

When (1) was checked against the formal system model, it verified to true in 0.90 s having visited 1648 states. Thus, the radiation therapy machine will never irradiate a patient if the human operator behaves normatively.

6.4. Analyzing system safety with up to one erroneous act

6.4.1. Translation

The EOFM instance was re-translated into SAL code, this time allowing for up to one erroneous act ($EMax=1$). The produced erroneous EOFM model (an example of which appears in Fig. 8) was 272 lines of XML code. Its formal representation was 739 lines of SAL code.

6.4.2. Verification

The specification in (1) was checked against this new system model. This produced a counterexample after 44.11 s indicating that, when the human operator performs up to one erroneous act, the specification in (1) can be violated.

To aid in the problem diagnosis, the counterexample was visualized using a technique documented by Bolton and Bass (2010b). In this approach, for each step in a counterexample, formal model variables are listed under categories based on their location in the architecture of the formal model (human–device interface, device automation, human mission, human task behavior, and operational environment). Changes in variable values from the previous step are highlighted. Further, the execution state of the EOFM task model is rendered using the EOFM visual notation for each step of the counterexample, where an activity in a rendered structure is color coded to indicate if it is *Ready*, *Executing*, or *Done*.

The counterexample illustrated one way that the specification violation could occur:

1. The model initialized with the interface in the edit state with no displayed data and the beam not ready; the beam power level not set; the spreader out of place; the beam fire state waiting; and the human mission indicating that the human operator should administer electron beam treatment.
2. When attempting to select the electron beam mode, the practitioner erroneously pressed 'X' instead of 'E' via a generated *Commission* activity in *aSelectXorE* from Fig. 7 (in the counterexample visualization, this is indicated by *ECount* changing from 0 to 1 and the rendering of the EOFM task structure showing that the generated *Commission* activity was *Executing*). This caused the interface to transition to the x-ray data confirmation state and display the x-ray treatment data. The spreader was also moved in place and the beam was set to the x-ray power level.
3. Because the incorrect data were displayed, the practitioner pressed up to return the interface to the edit mode.
4. The practitioner selected electron beam treatment mode by pressing the 'E' key. The interface transitioned to the electron beam data confirmation state and displayed the electron beam treatment data. The spreader was moved out of place and the beam prepared to transition to the electron beam power level (*XtoE* in Fig. 6(a)).

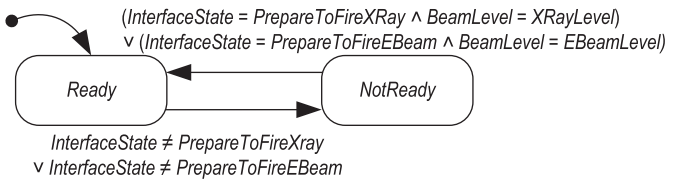

Fig. 9. State transition diagram depicting the modified formal model's human–device interface indications of the beams ready state.

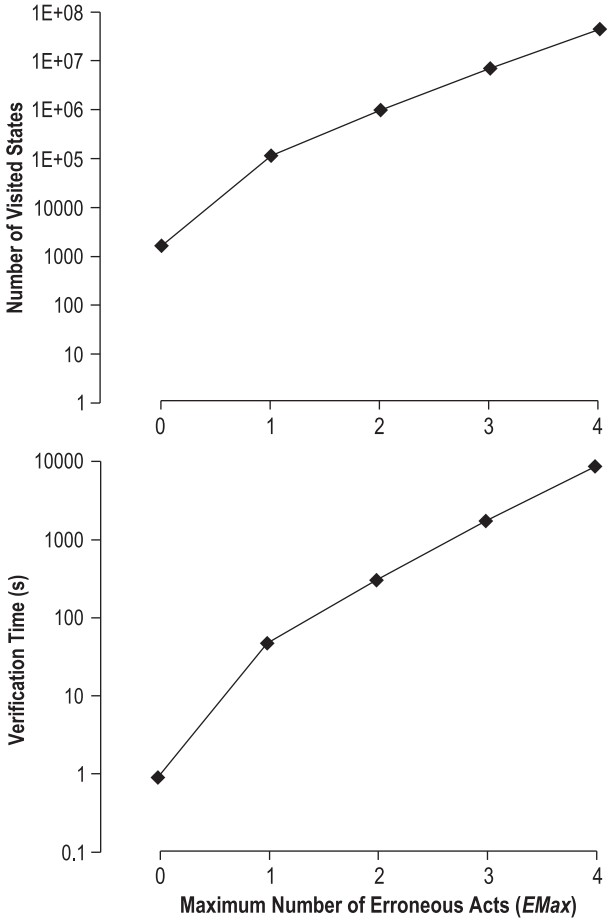

Fig. 10. Plot of the verification results (verification time in seconds and the number of visited states) vs. the maximum number of erroneous acts (*EMax*) for all successful verifications with the radiation therapy machine application.

5. The practitioner confirmed the treatment data by pressing enter and the interface transitioned to the electron beam's waiting to fire state.
6. The beam became ready.
7. The practitioner fired the beam by pressing 'B'. Because the beam power level had not yet transitioned to the electron beam power level, the beam fired at the x-ray power level with the spreader out of place.

6.4.3. Redesign

One way of eliminating this discovered system failure is by modifying when the beam becomes ready in the human–device interface. This can be done by adding the additional constraint that the beam will not transition to ready unless it is set to the correct power level (Fig. 9).

6.4.4. Re-verification

When this modified model was incorporated into the system model with the erroneous human behavior model ($EMax=1$) and checked against (1), it verified to true in 47.06 s having visited 114,708 states.

6.5. Analyzing system safety with two or more erroneous acts

6.5.1. Translation

Using the modified human–device interface design from the previous section, the EOFM instance was re-translated into SAL code three more times allowing for up to two ($EMax=2$; 455 lines of XML and 1028 lines of SAL code), three ($EMax=3$; 598 lines of XML and 1364 lines of SAL code), and four ($EMax=4$; 741 lines of XML and 1700 lines of SAL code) erroneous acts.

6.5.2. Verification

In all cases, when (1) was checked against the resulting system model, it verified to true (see Fig. 10 for verification statistics). Thus, with the redesigned human–device interface, the model indicates that the patient will never be irradiated even if the human operator performs up to four erroneous acts.

7. Discussion

The method presented here represents a novel contribution to model-driven design and analysis techniques that use formal verification to evaluate the role of human behavior in system safety. Using task analytic behavior models, phenotypical erroneous behavior generation, formal modeling, and model checking, the presented methods gives analysts the ability to use task analytic human behavior models to evaluate whether or not normative human behavior and/or potentially unanticipated variations from that behavior will or will not contribute to violations of system safety. Further, the nature of the presented technique allows the analyst to scale his or her evaluation to allow for the generation of a number of different erroneous behaviors. By using an EOFM task structure to replace every action in an instantiated EOFM, the method is capable of generating zero-order phenotypes of erroneous human action: omitting an action, skipping an action, re-performing a previously completed action, repeating the last performed action, or intruding an action. The use of the structure in each action, and the use of the *or_seq* decomposition operator (which allows one or more sub-activities to execute) in each instance of the structure allow multiple zero-order phenotypical erroneous acts to generate all of Hollnagel's first-order phenotypes except for time compression. The number of possible erroneous acts is constrained by a maximum and a counter preventing generated erroneous behavior from making the task behavior model unbounded. The erroneous behavior generation structure is represented in EOFM constructs and is thus compatible with the EOFM to SAL translator and counterexample visualizer.

While simple, the radiation therapy machine example illustrates how this process can be used to find potential system problems due to erroneous human behavior in HAI systems. The discovered problem is very similar to one found in the Therac-25 (Leveson and Turner, 1993). Similar problems have also resulted in injury and death with a number of modern radiation therapy machines (Bogdanich, 2010).

7.1. Excluding specific erroneous behaviors

The nature of model checking is such that an analyst may uncover a counterexample illustrating a system problem involving an erroneous human behavior neither interesting nor correctable. In this situation, the analyst may wish to rerun the analysis without considering a specific erroneous behavior.

In the current implementation, this can be achieved by modifying the specification to ensure that the unwanted behavior is never executed. This can be accomplished with the following specification:

$$(\mathbf{G}(ErroneousAct = Ready)) \Rightarrow (OriginalSpecification) \quad (2)$$

This asserts that, for the EOFM Action or Activity representing the erroneous act the analyst wishes to ignore (*ErroneousAct*) and the original desired specification (*OriginalSpecification*), the erroneous act never executing ($\mathbf{G}(ErroneousAct = Ready)$) will imply that the original specification will always be true.

For example, where *PressX* represents the action for erroneously pressing 'X' instead of 'E' when selecting the treatment mode, we can apply this to the un-corrected radiation therapy machine model with $EMax=1$ erroneous human behaviors as:

$$(\mathbf{G}(PressX = Ready)) \Rightarrow \left(\mathbf{G}\neg\left(\begin{array}{l} BeamFireState = Fired \\ \wedge\, BeamLevel = XRayPowerLevel \\ \wedge\, Spreader = OutOfPlace \end{array}\right)\right) \quad (3)$$

This specification verifies to true.

Note that (2) can be modified to exclude additional erroneous acts simply by adding them to the expression to the left of the implication operator.

Future work should investigate other means of allowing analysts to exclude specific erroneous acts or groups of them from formal verifications.

7.2. Statespace complexity

The fact that the complexity (and verification time) of the model increases exponentially with the maximum number of allowable erroneous acts indicates that analysts will be limited in the number of erroneous acts they can introduce in a formal verification. Even with the simple benchmark models with maxima of only eight erroneous acts, we were able to generate models that took longer than an hour to verify and exceeded the memory capacity (16 GB) of our analysis machine. These limitations are also seen in the radiation therapy application in the exponential increase seen in the statespace with each increase in

maximum number of erroneous acts (Fig. 10). That said, verifications with up to a maximum a four erroneous acts were performed with the application without coming close to reaching the analysis limit.

There may be possibilities for future improvements. Some of the increased complexity of the model containing erroneous behavior is associated with the fact that every activity or action incorporated into an instantiated EOFM task behavior model adds a variable (representing the activity's or action's execution state) with three possible values. Thus, the complexity of the model would likely decrease with a reduction in the number of additional activities and actions contained in the generative structure. One way of doing this would be to remove the multiple *Repetition* and *Commission* activities and allow single instances of them to execute multiple times. Implementing this with our current technology would require that the erroneous behavior generation structure not adhere to the EOFM's formal semantics (Bolton et al., 2011). Future work should investigate if such a solution improves model complexity and whether EOFM formal semantics should be extended to support this.

Another possibility for reducing the statespace of the model lies in eliminating the intermediary transitions of EOFM activity and action execution states. In the current implementation of the translator (Bolton et al., 2011), the formal model represents every transitional step of an activity's execution state as a single transition in the formal model. While this accurately adheres to the formal semantics of the language, the formal model requires a discrete state for every intermediary transition in the task model hierarchy. These intermediary transitions are of no consequence to the other elements of the system model, which are only concerned with the resulting human actions. Thus, the complexity of the formal task model representation could be reduced by decreasing the number of internal transitions required to traverse an instantiated EOFM's translated task structure (including the erroneous behavior generation structure) during execution. Future work should investigate the feasibility of this approach.

The reported benchmarks (Fig. 4) show that some decomposition operators scale better than others. It is not clear how often the less scalable decompositions will be used. However, every decomposition operator has been used in some application (Bolton, 2010, 2011; Bolton and Bass, 2009a,b, 2010a,b, in press; Bolton et al., 2011). Future work should investigate how the frequency with which different decomposition operators are used will impact the scalability of the method.

Even though the scalability of the current implementation could be improved, the radiation therapy example illustrates that significant insights into the safety of a system can be found with only a maximum of one erroneous act. As such, our method would likely be most effective if applied iteratively (as was done in this paper), where analysts start with a maximum of one erroneous act and progressively increase the maximum number of erroneous acts until the necessary insights are gained or the model exceeds the available computational resources.

7.3. Future extension of our method

Our method currently does not support Hollnagel's (Hollnagel, 1993) phenotypes for delaying, prematurely starting, or prematurely finishing an action. Future work should investigate if these types of erroneous behaviors can be incorporated into our analyses using timed automata (Henzinger et al., 1991).

The EOFM language supports the synchronous execution of actions, where a *sync* decomposition operator is used to specify that all of the actions in a given decomposition execute at the same time. Hollnagel's phenotypes assume that human behavior is executing sequentially, with no parallelism or synchronization between human actions. For this reason, the current method does not support the erroneous behavior generation for synchronized human actions and does not support synchronized erroneous actions within a given generative structure, such as a human performing two erroneous actions synchronously or performing the correct action synchronously with erroneous actions. Future work should investigate how parallelism could be incorporated into our erroneous human behavior generation structure, and investigate how this impacts the statespace complexity and verification time of models in which it is used.

For a large maximum number of erroneous acts, our approach will ultimately be capable of reproducing the more complicated and cognitively significant erroneous behavior patterns that have been explored by Basnyat and Palanque (2005), Palanque and Basnyat (2004), and Paternò and Santoro (2002). However, doing so will also generate a significant number of complex erroneous behaviors that are not as cognitively probable (see Reason, 1990). Future work should explore alternative erroneous behavior generation processes that might be capable of generating these types of patterns with EOFM and our analysis method.

If the model checker encounters a situation that the human task behavior model cannot address (it deadlocks), it will abandon it as a path of inquiry in the model. This is not realistic because, if real human operators encounter such situations, they may respond in a number of different ways. For example, they may attempt other means of achieving the task, they may restart the task, or they may abandon it and move on to other tasks. All of these behaviors could be viewed as erroneous since they do not conform to the task model. Thus, they should be included in formal verification analyses concerned with generating erroneous human behavior. Palanque et al. (2011) have modeled interruptions when evaluating systems with task analytic behavior models. Doherty et al. (2008) have investigated a 'resourced action' approach to modeling human goal directed behavior without formally articulated task models. Future work should investigate whether these

techniques could be adapted to model the human response to task deadlock in our infrastructure.

7.4. Comparison of existing techniques

As described in the introduction, there are tradeoffs in scope and purpose between the different methods that allow formal verification to be used to evaluate the safety of human-automation interactive systems. There are scalability tradeoffs as well. Analyses that only use a model of the human–device interface will likely scale better than the other presented techniques, but will explore many different human behaviors that may not be realistic or useful. Techniques that use erroneous behavior patterns that analysts must manually include in task analytic models will also likely scale better than the presented method, but they will not allow for the inclusion of unanticipated erroneous behavior. Analyses using cognitive models allow the impact of unanticipated (cognitively caused) erroneous behavior to be evaluated, but it is not clear how such analyses will scale compared to the method presented here.

To better understand the tradeoffs between techniques, future work should investigate how an infrastructure could be constructed to enable all of these approaches to be applied to a common system. Such an infrastructure would facilitate the comparison of the different techniques across many applications. This would allow analysts to determine what types of systems the different techniques are compatible with, how they scale relative to each other, and what types of problems they can discover.

It should be noted that, because of the similar modeling approaches, the method presented here should be capable of being used synergistically with the unconstrained and other task-model-based techniques that have been discussed. For example, an analyst can use unconstrained task behavior (a task model that allows a human operator to perform any action at any time) to evaluate system safety properties that must always be true no matter the human behavior. The analyst can incorporate a normative human task behavior model and verify that other system safety properties that depend on human behavior will be maintained. The analyst can then apply the technique presented in this paper to see if expected deviations from the task model (based on the phenotypes of erroneous action) will result in violations of these safety properties. Finally, the analyst can evaluate whether more complex erroneous behaviors impact system safety by incorporating the erroneous behavior patterns discussed by (Basnyat and Palanque, 2005; Palanque and Basnyat, 2004; Paternò and Santoro, 2002) into the normative task behavior model. It is not clear how the methods that utilize cognitive architectures could be made to work synergistically with these other techniques. This should be explored in future research.

Acknowledgments

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), 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

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

Aït-Ameur, Y., Baron, M., 2006. Formal and experimental validation approaches in HCI systems design based on a shared event B model. International Journal on Software Tools for Technology Transfer 8 (6), 547–563.

Aït-Ameur, Y., Baron, M., Girard, P., 2003. Formal validation of HCI user tasks. In: Proceedings of the International Conference on Software Engineering Research and Practice. CSREA Press, Las Vegas, pp. 732–738.

BASI, 1998. Advanced Technology Aircraft Safety Survey Report. Technical Report. Department of Transport and Regional Dedvelopment, Bureau of Air Safety Investigation, Civic Square.

Basnyat, S., Palanque, P., 2005. A task pattern approach to incorporate user deviation in task models. In: Proceedings of the first ADVISES Young Researchers Workshop. Ris ϕ National Laboratory, Roskilde, pp. 10–19.

Basnyat, S., Palanque, P., Schupp, B., Wright, P., 2007. Formal socio-technical barrier modelling for safety-critical interactive systems design. Safety Science 45 (5), 545–565.

Basnyat, S., Palanque, P., Bernhaupt, R., Poupart, E., 2008. 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. Taylor and Francis Group, London, pp. CD–ROM.

Bastide, R., Basnyat, S., 2007. Error patterns: systematic investigation of deviations in task models. In: Task Models and Diagrams for Users Interface Design. Springer, Berlin, pp. 109–121.

Basuki, T.A., Cerone, A., Griesmayer, A., Schlatte, R., 2009. Model-checking user behaviour using interacting components. Formal Aspects of Computing, 1–18.

Berstel, J., Reghizzi, S.C., Roussel, G., Pietro, P.S., 2005. A scalable formal method for design and automatic checking of user interfaces. ACM Transactions on Software Engineering and Methodology 14, 124–167.

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

Blandford, A., Butterworth, R., Good, J., 1997. 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. Springer, Berlin, pp. 45–60.

Bogdanich, W., 2010. The radiation boom: radiation offers new cures, and ways to do harm. The New York Times 23 (January), 23–27.

Bolton, M.L., 2010. Using task analytic behavior modeling, erroneous human behavior generation, and formal methods to evaluate the role

of human-automation interaction in system failure. Ph.D. Thesis. University of Virginia, Charlottesville.

Bolton, M.L., 2011. Validating human-device interfaces with model checking and temporal logic properties automatically generated from task analytic models. In: Proceedings of the 20th Behavior Representation in Modeling and Simulation Conference. The BRIMS Society, Sundance, pp. 130–137.

Bolton, M.L., Bass, E.J., 2009a. Enhanced operator function model: a generic human task behavior modeling language. In: Proceedings of the IEEE International Conference on Systems Man and Cybernetics. IEEE, Piscataway, pp. 2983–2990.

Bolton, M.L., Bass, E.J., 2009. A method for the formal verification of human interactive systems. In: Proceedings of the 53rd Annual Meeting of the Human Factors and Ergonomics Society. HFES, Santa Monica, pp. 764–768.

Bolton, M.L., Bass, E.J., 2010a. Formally verifying human-automation interaction as part of a system model: limitations and tradeoffs. Innovations in Systems and Software Engineering: A NASA Journal 6 (3), 219–231.

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

Bolton, M.L., Bass, E.J. Using model checking to explore checklist-guided pilot behavior. International Journal of Aviation Psychology, in press.

Bolton, M.L., Bass, E.J., Siminiceanu, R.I., 2008. Using formal methods to predict human error and system failures. In: Proceedings of the 2nd International Conference on Applied Human Factors and Ergonomics. Applied Human Factors and Ergonomics International, Las Vegas, pp. CD–ROM.

Bolton, M.L., Siminiceanu, R.I., Bass, E.J., 2011. A systematic approach to model checking human-automation interaction using task-analytic models. IEEE Transactions on Systems, Man, and Cybernetics, Part A 41 (5), 961–976.

Bredereke, J., Lankenau, A., 2005. Safety-relevant mode confusions–modelling and reducing them. Reliability Engineering and System Safety 88 (3), 229–245.

Buth, B., 2004. Analyzing mode confusion: an approach using FDR2. In: Proceeding of the 23rd International Conference on Computer Safety, Reliability, and Security. Springer, Berlin, pp. 101–114.

Butterworth, R., Blandford, A., Duke, D., 1998. 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. Springer, Berlin, pp. 87–101.

Butterworth, R., Blandford, A., Duke, D., 2000. Demonstrating the cognitive plausibility of interactive system specifications. Formal Aspects of Computing 12 (4), 237–259.

Byrne, M.D., Bovair, S., 1997. A working memory model of a common procedural error. Cognitive Science 21 (1), 31–61.

Campos, J.C., 2003. Using task knowledge to guide interactor specifications analysis. In: In Proceedings of the 10th International Workshop on Interactive Systems, Design, Specification, and Verification. Springer, Berlin, pp. 171–186.

Campos, J.C., Harrison, M., 1997. Formally verifying interactive systems: a review. In: Proceedings of the Fourth International Eurographics Workshop on the Design, Specification, and Verification of Interactive Systems. Springer, Berlin, pp. 109–124.

Campos, J.C., Harrison, M.D., 2001. Model checking interactor specifications. Automated Software Engineering 8 (3), 275–310.

Campos, J.C., Harrison, M.D., 2008. 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. Springer, Berlin, pp. 72–85.

Campos, J.C., Harrison, M.D., 2009. Interaction engineering using the ivy tool. In: Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, New York, pp. 35–44.

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

Clarke, E.M., Grumberg, O., Peled, D.A., 1999. Model Checking. MIT Press, Cambridge.

Combéfis, S., Giannakopoulou, D., Pecheur, C., Feary, M., 2011. Learning system abstractions for human operators. In: Proceedings of the 2011 International Workshop on Machine Learning Technologies in Software Engineering. ACM, New York, pp. 3–10.

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

Curzon, P., Blandford, A., 2004. Formally justifying user-centered design rules: a case study on post-completion errors. In: Proceedings of the 4th International Conference on Integrated Formal Methods. Springer, Berlin, pp. 461–480.

Curzon, P., Rukšėnas, R., Blandford, A., 2007. An approach to formal verification of human-computer interaction. Formal Aspects of Computing 19 (4), 513–550.

De Moura, L., Owre, S., Shankar, N., 2003. The SAL language manual. Tech. Rep. CSL-01-01, Computer Science Laboratory, SRI International, Menlo Park.

Degani, A., 2004. Taming HAL: Designing Interfaces Beyond 2001. Macmillan, New York.

Degani, A., Gellatly, A., Heymann, M., 2011. Hmi aspects of automotive climate control systems. In: Proceeding of the IEEE International Conference on Systems, Man, and Cybernetics. IEEE, Piscataway, pp. 1795–1800.

Degani, A., Heymann, M., 2002. Formal verification of human-automation interaction. Human Factors 44 (1), 28–43.

Dix, A., Ghazali, M., Gill, S., Hare, J., Ramduny-Ellis, D., 2008. Physigrams: modelling devices for natural interaction. Formal Aspects of Computing, 1–29.

Doherty, G., Campos, J.C., Harrison, M.D., 2008. Resources for situated actions. In: Proceedings of the 15th International Workshop on Interactive Systems. Design, Specification, and Verification. Springer, Berlin, pp. 194–207.

Duan, L., Hofer, A., Hussmann, H., 2010. Model-based testing of infotainment systems on the basis of a graphical human-machine interface. In: Proceedings of the 2nd International Conference on Advances in System Testing and Validation Lifecycle. IEEE, Piscataway, pp. 5–9.

Dwyer, M.B., Carr, V., Hines, L., 1997. Model checking graphical user interfaces using abstractions. In: Proceedings of the Sixth European Software Engineering Conference. Springer, New York, pp. 244–261.

Dwyer, M.B., Robby, Tkachuk, O., Visser, W., 2004. Analyzing interaction orderings with model checking. In: Proceedings of the 19th IEEE International Conference on Automated Software Engineering. IEEE Computer Society, Los Alamitos, pp. 154–163.

FAA Human Factors Team, 1996. Federal aviation administration human factors team report on: the interfaces between flightcrews and modern flight deck systems. Tech. rep., Federal Aviation Administration, Washington, DC.

Fields, R.E., 2001. Analysis of erroneous actions in the design of critical systems. Ph.D. Thesis. University of York, York.

Giannakopoulou, D., Rungta, N., Feary, M., 2011. Automated test case generation for an autopilot requirement prototype. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. IEEE, Piscataway, pp. 1825–1830.

Giese, M., Mistrzyk, T., Pfau, A., Szwillus, G., von Detten, M., 2008. AMBOSS: a task modeling approach for safety-critical systems. In: Proceedings of the Second International Conference on Human-Centered Software Engineering. Springer, Berlin, pp. 98–109.

Gimblett, A., Thimbleby, H., 2010. User interface model discovery: Towards a generic approach. In: Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, New York, pp. 145–154.

Gunter, E.L., Yasmeen, A., Gunter, C.A., Nguyen, A., 2009. Specifying and analyzing workflows for automated identification and data capture. In: Proceedings of the 42nd Hawaii International Conference on System Sciences. IEEE Computer Society, Los Alatimos, pp. 1–11.

Harel, D., 1987. Statecharts: a visual formalism for complex systems. Science of Computer Programming 8 (3), 231–274.

Harrison, M.D., Duke, D., 1995. A review of formalisms for describing interactive behaviour. In: Proceedings of the Workshop on Software Engineering and Human-Computer Interaction. Springer, London, pp. 49–75.

Hartson, H.R., Siochi, A.C., Hix, D., 1990. The UAN: a user-oriented representation for direct manipulation interface designs. ACM Transactions on Information Systems 8 (3), 181–203.

Henzinger, T.A., Manna, Z., Pnueli, A., 1991. Timed transition systems. In: Proceedings of the REX Workshop. Springer, Berlin, pp. 226–251.

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

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

Hughes, D., Dornheim, M.A., 1995. Accidents direct focus on cockpit automation. Aviation Week and Space Technology 142 (5), 52–54.

Hussmann, H., Meixner, G., Detlef, Z., 2011. Model-Driven Development of Advanced User Interfaces. Springer, Berlin.

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

John, B.E., Kieras, D.E., 1996. Using GOMS for user interface design and evaluation: which technique? ACM Transactions Computer-Human Interaction 3 (4), 287–319.

Johnson, C., Holloway, C., 2004. On the over-emphasis of human 'error' as a cause of aviation accidents: 'systemic failures' and 'human error' in US NTSB and Canadian TSB aviation reports 1996–2003. In: Proceedings of the 22nd International System Safety Conference. International Systems Safety Society, Unionville, pp. CD–ROM.

Jones, P.M., 1997. Human error and its amelioration. In: Handbook of systems engineering and management. Wiley, Malden, pp. 687–702.

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

Kirwan, B., Ainsworth, L.K., 1992. A Guide to Task Analysis. Taylor and Francis, London.

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

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

Leveson, N.G., Turner, C.S., 1993. An investigation of the therac-25 accidents. Computer 26 (7), 18–41.

Martinie, C., Palanque, P., Barboni, E., Ragosta, M., 2011. Task-model based assessment of automation levels: application to space ground segments. In: Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics. IEEE, Piscataway, pp. 3267–3273.

Masci, P., Curzon, P., Blandford, A., Furniss, D., 2011. Modelling distributed cognition systems in PVS. In: Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems. EASST, Potsdam.

Mitchell, C.M., Miller, R.A., 1986. A discrete control model of operator function: a methodology for information display design. IEEE Transactions on Systems Man Cybernetics Part A: Systems and Humans 16 (3), 343–357.

Norman, D.A., 1988. The Psychology of Everyday Things. Basic Books, New York.

Norman, D.A., 1990. The problem with automation: inappropriate feedback and interaction, not over-automation. Philosophical Transactions of the Royal Society of London. Series B Biological Sciences 327, 585–593.

O'Hara, J.M., Higgins, J.C., Brown, W.S., Fink, R., Persensky, J., Lewis, P., Kramer, J., Szabo, A., Boggi, M.A., 2008. Human factors considerations with respect to emerging technology in nuclear power plants. Technical Report NUREG/CR-6947. United States Nuclear Regulatory Commission, Washington, DC.

Palanque, P., Basnyat, S., 2004. 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. Kluwer Academic Publisher, Norwell, pp. 109–130.

Palanque, P., Winckler, M., Martinie, C., 2011. A formal model-based approach for designing interruptions-tolerant advanced user interfaces. In: Hussmann, H., Meixner, G., Zuehlke, D. (Eds.), Model-Driven Development of Advanced User Interfaces. Springer, Berlin, pp. 143–169.

Palanque, P., Bastide, R., Senges, V., 1996. 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. Chapman and Hall, Ltd., London, pp. 189–212.

Parnas, D.L., 1969. On the use of transition diagrams in the design of a user interface for an interactive computer system. In: Proceedings of the 24th National ACM Conference. ACM, New York, pp. 379–385.

Paternò, F., 1997. Formal reasoning about dialogue properties with automatic support. Interacting with Computers 9 (2), 173–196.

Paternò, F., Mancini, C., Meniconi, S., 1997. Concurtasktrees: a diagrammatic notation for specifying task models. In: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction. Chapman and Hall, Ltd., London, pp. 362–369.

Paternò, F., Santoro, C., 2001. 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. Springer, Berlin, pp. 135–150.

Paternò, F., Santoro, C., 2002. Preventing user errors by systematic analysis of deviations from the system task model. International Journal of Human-Computer Studies 56 (2), 225–245.

Paternò, F., Santoro, C., Tahmassebi, S., 1998. 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. Springer, Vienna, pp. 71–86.

Reason, J., 1990. Human Error. Cambridge University Press, New York.

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

Rukšenas, R., Back, J., Curzon, P., Blandford, A., 2009. Verification-guided modelling of salience and cognitive load. Formal Aspects of Computing 21 (6), 541–569.

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

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

Sarter, N.B., Woods, D.D., 1995. How in the world did we ever get into that mode? Mode error and awareness in supervisory control. Human Factors 37 (1), 5–19.

Sherry, L., Ward, J., 1995. A formalism for the specification of operationally embedded reactive systems. In: Proceeding of the 14th Digital Avionics Systems Conference. IEEE, Piscataway, pp. 416–421.

Thimbleby, H., Gow, J., 2007a. Applying graph theory to interaction design. In: Proceedings of the 2007 Engineering Interactive Systems Conference. Springer, Berlin, pp. 501–519.

Thimbleby, H., Gow, J., 2007b. Applying graph theory to interaction design. In: Proceedings of Engineering Interactive Systems 2007. Springer, Berlin, pp. 501–519.

Thomas, M., 1994. The story of the therac-25 in lotos. High Integrity Systems 1 (1), 3–15.

Thurman, D.A., Chappell, A.R., Mitchell, C.M., 1998. An enhanced architecture for OFMspert: a domain-independent system for intent inferencing. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. IEEE, Piscataway, pp. 955–960.

Wheeler, P.H., 2007. Aspects of automation mode confusion. Master's Thesis. Massachusetts Institute of Technology, Cambridge.

Wing, J.M., 1990. A specifier's introduction to formal methods. Computer 23 (9), 8 10–22, 24.

Young, R.M., Green, T.R.G., Simon, T., 1989. Programmable user models for predictive evaluation of interface designs. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, New York, pp. 15–19.