Int. J. Human-Computer Studies 108 (2017) 105–121

Contents lists available at ScienceDirect

International Journal of Human-Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

A task-based taxonomy of erroneous human behavior

Matthew L. Bolton

University at Buffalo, State University of New York, Department of Industrial and Systems Engineering, Buffalo, NY, USA

ARTICLE INFO

Keywords:
Human error
Erroneous human behavior
Task analysis
System safety
Formal methods

ABSTRACT

Unexpected, erroneous human interactions often contribute to failures in complex systems. Human factors engineers and researchers have developed taxonomies that allow engineers, designers, and practitioners to think about and model erroneous behavior to improve the safety of human-interactive systems. However, the two leading erroneous behavior taxonomies are based on incompatible phenomenological and genotypical perspectives. Further, neither of these are formulated in terms of task analytic methods, analysis and modeling techniques human factors engineers use for documenting how humans normatively achieve goals when interacting with a system. In this work, we introduce a new erroneous human behavior taxonomy based on where and how human behavior diverges from task analytic models of human behavior. By describing where a human diverges from a normative task, and by identifying what information the human failed to properly attend to that produced the divergence, this taxonomy seeks to unify the phenomenological and genotypical perspectives. We describe the theory behind this taxonomy and the different erroneous behavior classifications that result from it. We then show how it is compatible with the leading phenomenological and genotypical taxonomies. Finally, we discuss the implications of this new taxonomy and avenues of future research.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

In complex systems that depend on human behavior, unexpected erroneous human interactions often contribute to system failures (Hollnagel, 1993; [Perrow, 1999]; Reason, 1990; Sheridan and Parasuraman, 2005). This problem affects nearly every facet of our society. It plays a role in between 44,000 and 98,000 deaths and more than 1,000,000 injuries a year in medicine (Kohn et al., 2000); 74% of general aviation accidents and 50% of commercial aviation accidents (Kebabjian, 2016; Kenny, 2015); 90% of automobile accidents (NHTSA, 2008); a third of unmanned aerial vehicle mishaps (Manning et al., 2004); and many incidents of fratricide in military operations (Office of Technology Assessment, 1993).

Erroneous behavior is often a product of poorly designed human interaction and is thus enabled by the tasks of the system (Hollnagel, 1993; Reason, 1990; Sheridan and Parasuraman, 2005). Taxonomies have been developed to give analysts and engineers ways of thinking about, classifying, and modeling erroneous human behavior (Hollnagel and Marsden, 1996; Jones, 1997). The two leading general taxonomies are Reason's (1990) Generic Error Modeling System (GEMS) and Hollnagel's (1993) phenotypes of erroneous action. GEMS attempts to explain why, cognitively, erroneous acts occur (their genotypes). The phenotypes of erroneous action describe how erroneous behaviors observably manifest as deviations from a plan of action. Both have proven to be extremely influential in the extended human factors literature, though they are useful in different contexts. Further, neither specifically address where in a task the human operator diverges. In the work presented here, we have developed a taxonomy that classifies erroneous human behavior based on where and how it deviates from a normative human task. This is a significant contribution because it allows engineers and analysts to contextualize erroneous behavior using task-analytic concepts, which are a cornerstone of human factors engineering. It also connects the cognitive reasons errors occur with the actual observable phenotypes of error, reconciling GEMS and Hollnagel's phenotypes. Below, we discuss the background necessary for understanding our approach. We then present our taxonomy. Following this, we show that our approach can be used to bridge the gap between attentional slips in GEMS and the phenotypes of erroneous behavior. We do this by demonstrating that our new system achieves coverage with respect to these other taxonomies: that it is able to account for all of the erroneous behaviors classified in these taxonomies. We ultimately discuss our results, the implications of the taxonomy in human factors engineering, and avenues of future research.

2. Review of the relevant literature

Task analysis and erroneous human behavior are relevant to this work and thus both are discussed below.

E-mail address: mbolton@buffalo.edu

http://dx.doi.org/10.1016/j.ijhcs.2017.06.006
Received 23 August 2016; Received in revised form 12 June 2017; Accepted 27 June 2017
Available online 28 June 2017
1071-5819/© 2017 Elsevier Ltd. All rights reserved.

2.1. Task analysis and task analytic models

Task analysis is a systematic process human factors engineers use to describe the ways human operators normatively achieve goals with a system (Kirwan and Ainsworth, 1992; Schraagen et al., 2000). This is most commonly documented using a task analytic model. Such a model is a collection of individual tasks, where each is a hierarchy of goal-directed activities that decompose into other activities and (at the lowest level) actions. Strategic knowledge (condition logic) and operators control when and how activities can execute in relation to each other and the operational environment. Task analytic models are some of the most successful technologies developed by human factors researchers. They are critical to human-centered design (where human-machine interfaces are designed to support human operator tasks; Cooley, 2000). They can be used to generate human-machine interfaces in model-based design and analysis (Li et al., 2015, 2017). They are used in formal verifications analysis of human-automation interaction (Bolton et al., 2013) (where properties are proven about models of human-interactive systems). They are important in human-reliability analyzes (Hollnagel, 1998). Finally, they are employed in accident and event analysis (Doytchev and Szwillus, 2009).

There are a number of different ways to represent task analytic models. Because this work is concerned with deviations from task models, it is important to be able to reason about the execution state of the task. Researchers have developed task analytic modeling systems that allow task behavior to be reasoned about with mathematical precision. These include ConcurTaskTrees (CTT; Paternò et al. 1997), its extension Hamster (Martinie De Almeida et al., 2013), AMBOSS (Giese et al., 2008), and the Enhanced Operator Function Model (EOFM; Bolton et al. 2011). EOFM is expressive, platform independent, feature rich, has a well-documented formal semantics (Bolton et al., 2011, 2016), and has been used in a number of human factors analyzes (Bolton and Bass, 2017). Thus, it will be used on the basis for most of the discussion presented in this paper.

2.2. Erroneous human behavior

There are a number of different ways to classify and model erroneous human behavior (Hollnagel and Marsden, 1996; Jones, 1997). The two generic taxonomies that have seen the most widespread use are Hollnagel's phenotypes of erroneous action (1993) and Reason's GEMS (1990).

2.2.1. The phenotypes of erroneous action

Hollnagel (1993) classified erroneous human behaviors based on their phenotype: how erroneous behavior observably deviates from a normative plan of actions. 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 action in a plan. The "zero-order" phenotypes are classified as shown at the top of Table 1. These serve as the building blocks for additional, "first-order," phenotypes: phenotypes that can be detected by observing multiple zero-order phenotypes (see the bottom of Table 1). Hollnagel (1993) also discussed "second-order" phenotypes, which can be detected by observing multiple first-order phenotypes. However, he did not explicitly describe the specific phenotypes of this level.

2.2.2. The generic error modeling system

In contrast to Hollnagel, Reason (1990) classified erroneous behaviors based on their cognitive causes, their genotypes. Reason identified three types of errors based on Rasmussen's SRK (Skills, Rules, Knowledge) framework (Rasmussen, 1983). That is, errors could occur at any of the three levels of human information processing. Skill-based failures (slips) occur when the human operator knows how to perform a task and intends to do it correctly. However, the person fails execute the plan because of problems with attention. Rule-based failures (mistakes) occur when a human operator performs a rule-based behavior that does not achieve the intended outcome because he or she did not apply a rule correctly or had an inadequate plan of action. Knowledge-based failures (mistakes) happen when the human lacks the knowledge to formulate a proper plan of action for a given situation. The distinction between slips and mistakes largely comes down to the fact that the latter occurs when the human operator does not know how to properly perform a task. Thus, for work where there is a clear task model, there is a basic assumption that the human operator is properly trained and practicable in the task. Thus, slips are the most relevant to the work discussed here.

Table 1
Hollnagel's (1993) phenotypes of erroneous action.

Phenotype	Description
Zero-order phenotypes	
Premature Start	Starting an action too early.
Delayed Start	Starting an action too late.
Premature Finish	Finishing an action too soon.
Delayed Completion	Finishing an action too late.
Omission	Not performing an action.
Jump Forward	Performing an action that should be done later.
Jump Backward	Performing a previously performed action.
Repetition	Repeating the last action performed.
Intrusion	Performing an unplanned action.
First-order phenotypes	
Spurious Intrusion	Performing a sequence of unplanned actions via multiple zero-order intrusions.
Jump / Skip	Actions are skipped in either the forward or backward direction.
Place Losing	Planned actions are performed in an arbitrary order via multiple skips and jumps.
Recovery	Performing previously omitted actions via multiple jumps.
Side Tracking	Replacing one part of a plan with another.
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.
Time Compression	Multiple premature starts and/or premature finishes.

Within the slip error type, Reason (1990) identified different failure modes to explain the ways that slips can manifest. First, there are failure modes associated with inattention (someone failing to attend to something). A *double capture slip* (commonly called a capture error) occurs when a human's attention is captured by something and thus does something different (usually something similar that is more well-rehearsed or familiar) from what should be done in his or her task. An *omission following an interruption* can occur as a result of a person not attending to the task following an external event. *Reduced intentionality* takes place when the human loses track of his or her intentions in the middle of a task. *Perceptual confusion* occurs in tasks where somebody performs the correct action on the wrong target due to similarities between the wrong target and the correct target. *Interference errors* happen when the human has more than one active or relevant task he or she should or could be performing and the separate tasks become inappropriately blended. Slips can also occur because of over attention (mistimed attentional checks) that can cause a human to perform an *omission* (not doing something), a *repetition* (repeat something already done), or a *reversal* (reversing the order of steps in a sequence). The distinctions between these failure modes are fuzzy, largely due to the informal nature of GEMS. This, however, does not negate the usefulness of this taxonomy.

2.2.3. Comparison

Both models of erroneous behavior have shown themselves to be useful (Jones, 1997). Hollnagel's phenotypes have been predominately used for activities such as detecting the presence of human errors in monitors (Hollnagel, 1993), methods for generating and/or exploring erroneous behavior in model-based analyzes with defined normative plans of action (Bastide and Basnyat, 2007; Bolton and Bass, 2008; Bolton et al.,

2012; Fields, 2001; Jones, 1997), and techniques such as forcing functions (Norman, 1988) for ensuring that humans remain on task. Conversely, Reason's GEMS has found more use in its ability to help system analysts and designers understand what causes erroneous behavior so that the system can be designed to support the level of work (skills, rules, and/or knowledge) the human is performing and to avoid cognitive conditions that facilitate the associated errors (Jones, 1997). For this purpose, Reason's taxonomy has also been adapted to specialized domains such as medicine so that its insights can be felt there (Zhang et al., 2004). Both phenomenological and genotypical perspectives can be used together as they are in Hollnagel's "Phenotype Genotype Classification Scheme" (Hollnagel, 1998). However, such perspectives simply use both classifications of erroneous behavior and do not account for the connection between them.

Both taxonomies can be related to task models, but in different ways. The phenotypes of erroneous action assume a normative plan of actions that is being observed. A task model is a normative plan. In GEMS, slips are presumed to occur when the humans have knowledge about how to perform activities. Such knowledge can be represented as a task model. Given the importance of task analysis in human factors engineering, it is surprising that neither of the taxonomies explain *where* in a task erroneous behaviors originate.

2.3. Erroneous behavior and task analytic models

Attempts have been made to contextualize erroneous behavior in task models. Paternò and Santoro (2002) explored different ways that deviations from task models could occur with CTT models. Similarly, the THEA (Technique for Human Error Assessment) system (Fields et al., 1997; Pocock et al., 2001a, 2001b) explored systematic ways that behavior could deviate from tasks based on Norman's execution evaluation cycle model of human information processing (Norman, 1988). These approaches are based on a keyword-guided brainstorming system for thinking about erroneous human behavior possibilities similar to hazard and operability study (HAZOP; Dunjó et al. 2010; Lawley 1974). Fields (2001) and Bastide and Basnyat (2007) identified specific patterns of human task behavior that could be manually inserted into task models to replicate erroneous behavior in simulation and formal verification analyzes. Bastide and Basnyat (2007) defined a pattern-based language and two specific "error patterns:" one for a repetition of behaviors and one for a post-completion error (an erroneous behavior condition where a human accomplishes their primary goal and then fails to perform tasks that must follow the completion of the primary goal; Byrne and Bovair 1997). Fields (2001) introduced patterns capable of replicating all of Hollnagel's (1993) phenotypes of erroneous action. Bolton et al. (2012) and Bolton and Bass (2013) introduced two different ways of automatically generating erroneous human behavior in executable task models. In the first (Bolton et al., 2012), actions at the bottom of the task hierarchy are replaced with task structures capable of generating all of Hollnagel's zero-order phenotypes and, through combination, all of his higher order phenotypes. In the second (Bolton and Bass, 2013), the models assume humans can fail to attend to environmental conditions asserted in task model strategic knowledge in situations that can make them repeat, omit, or commit activity behaviors erroneously. As such, this second method was capable of generating some of the slip behaviors in GEMS (1990). All of these methods have been useful in various analyzes, evaluation, and verification contexts. However, none are (nor do they aspire to be) full-fledged taxonomies. As such, none fully enumerates the connection between the observable manifestation of the erroneous behavior and the cognitive cause for the behavior.

3. A task-based taxonomy of erroneous human behavior

Task models are widely used in human factors engineering, encapsulate normative human behavior, and describe what environmental and task conditions the human should be attending to. Thus, deviations from task models are appropriate for classifying erroneous human behavior. Further, the discussion above has shown that both task models and deviations from task models are formalizable (can have a specific mathematical definition). As such, an erroneous behavior taxonomy based on deviations from task analytic models can be formal. This is advantageous because each classification within the taxonomy can have a precise, unambiguous classification.

Below we formulate our erroneous human behavior taxonomy that classifies erroneous behaviors formally based on where in a normative task model deviations occur. We then show that this taxonomy covers the erroneous behaviors classified in Hollnagel's phenotypes (1993) of erroneous action and slips from Reason's GEMS (1990). However, to be able to formulate our taxonomy, we need a formal description of a task model to employ as the basis for describing deviations. For this, we use the EOFM, which is described first. Note that the EOFM is appropriate for this work because it is an expressive task analytic modeling language. It also has formal semantics, which provide an unambiguous, mathematical description of how task models execute. These are used as the basis for the taxonomy.

Table 2
Decomposition operators.

Operator	Description
optor_seq	Zero or more of the activities or actions in the decomposition must execute in any order one at a time.
optor_par	Zero or more of the activities or actions in the decomposition must execute in any order and can execute in parallel.
or_seq	One or more of the activities or actions in the decomposition must execute in any order one at a time.
or_par	One or more of the activities or actions in the decomposition must execute in any order and can execute in parallel.
and_seq	All of the activities or actions in the decomposition must execute in any order one at a time.
and_par	All of the activities or actions in the decomposition must execute in any order and can execute in parallel.
xor	Exactly one activity or action in the decomposition executes.
ord	All activities or actions must execute in the order they appear in the decomposition.
sync	All actions in the decomposition must execute synchronously.

3.1. The Enhanced Operator Function Model (EOFM)

EOFM (Bolton et al., 2011) is a formal task analytic modeling language derived from the Operator Function Model (OFM; Mitchell and Miller 1986; Thurman et al. 1998). The language is XML-based and allows for the modeling of human behavior, either individuals or groups, as an input/output system. Inputs may come from other elements of the system like human-device interfaces or the environment. Output variables are human actions. The operators' task models describe how human actions may be generated based on input and local variables (representing perceptual or cognitive processes).

EOFMs are hierarchical in that they are composed of goal-driven activities that decompose into sub-activities and, at the bottom of the hierarchy, atomic actions. EOFMs can express strategic knowledge explicitly as Boolean expressions using input and local variables that assert what must be true for them to start executing (*Preconditions*), repeat (*RepeatConditions*), or complete (*CompletionConditions*). 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). Table 2 shows all of the decomposition operators currently supported by EOFM.

Observable, atomic human actions or internal (cognitive or perceptual) actions exist at the bottom of the task model hierarchy. Observable actions have three possible behaviors: *AutoReset* actions happen as a sin-

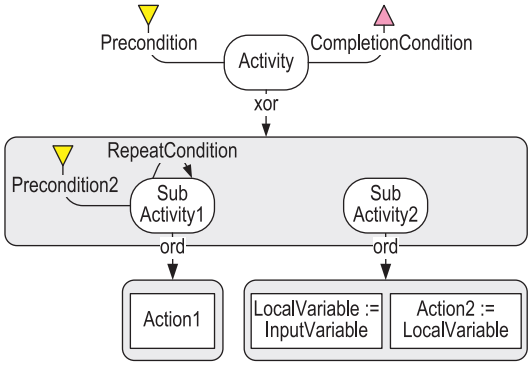

Fig. 1. An example of a visualized EOFM task. In this, a top activity (aActivity) decomposes into two sub-activities (SubActivity1 and SubActvity2). SubActivity1 and SubActvity2 each decompose into actions. Action1 represents an *AutoReset* action. LocalVariable represents a local variable assignment action, where the local variable is assigned the value of an input variable *InputVariable*. Action2 is an action with *SetValue* behavior that commits the values stored in *LocalVariable* when the action is performed.

gle atomic event; *Toggle* actions switch between occurring and not occurring whenever the action is performed; and *SetValue* actions convey a value that is more complex than simple occurrence or non-occurrence. Non-observable actions allow internal behaviors to be represented as the assignment of values to local variables. This can be used to represent the human operator remembering something or some other changes in cognitive or perceptual state.

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 an arrow labeled with the decomposition operator. The arrow points to a rounded rectangle containing the decomposed activities or actions. Strategic knowledge conditions are triangles and/or arrows connected to the activity that they constrain. These are labeled with the Boolean logic of the condition. A *Precondition* is a yellow, downward-pointing triangle; a *CompletionCondition* is a magenta, upward-pointing triangle; and a *RepeatCondition* is an arrow recursively pointing to the activity.

EOFMs have formal semantics (Bolton et al., 2011, 2016). This gives models represented with it unambiguous, mathematical descriptions of how they execute. Every activity and action is treated as a state machine (Fig. 2) that transitions between three execution states: *Ready* (waiting to execute), *Executing*, and *Done*. An activity or action starts in the *Ready* state. It transitions between states based on whether or not the specific Boolean conditions on the labeled transitions (Fig. 2) are true.

The strategic knowledge conditions of an activity (*Preconditions*, *RepeatConditions*, and *CompletionConditions*) are used to partially describe when these transitions can occur. However, three additional implicit conditions are also required. These assert whether an activity can start, end, or reset based on the given activity's or action's position in the task. Specifically, a *StartCondition* indicates if an activity can start executing based on the execution states of its parent, its parent's decomposition operator, and its siblings (activities or actions in the same decomposition). An *EndCondition* indicates if an activity or action can end execution based on the execution state of its children (activities or actions the activity decomposes into) and its decomposition operator. Since an action has no children, its *EndCondition* is true when the action has been properly executed. Finally, a *Reset* condition indicates when an activity or action can return to the *Ready* execution state.

For any activity or action in a decomposition, a *StartCondition* has two conjuncts:

$$(parent.state = Executing) \wedge \left(\bigwedge_{\forall siblings\ s} (s.state \neq Executing) \right). \quad (1)$$

If the parent's decomposition operator has a parallel modality, the second conjunct is eliminated. If the parent's decomposition operator is ord, the second conjunct imposes additional restrictions only on the previous sibling in the decomposition order: $(prev\_sibling.state = Done)$. If it is the xor decomposition operator, the second conjunct is modified to enforce the condition that no other sibling can execute after one has finished:

$$\bigwedge_{\forall siblings\ s} (s.state = Ready). \quad (2)$$

An activity without a parent (a top-level activity) will eliminate the first conjunct. Top-level activities treat each other as siblings in the formulation of the second conjunct with an assumed and_seq relationship. All other activities are treated as if they are in an and_par relationship and are thus not considered in the formulation of the *StartCondition*.

An *EndCondition* is comprised of two conjuncts that relate to the activity's children. Since an action has no children, an action's *EndCondition* defaults to true. The first conjunct asserts that the execution states of the activity's children satisfy the activity's decomposition operator. The second asserts that none of the children are *Executing*. This is generically expressed as:

$$\left(\bigoplus_{\forall subacts\ c} (c.state = Done) \right) \wedge \left(\bigwedge_{\forall subacts\ c} (c.state \neq Executing) \right). \quad (3)$$

In the first conjunct, $\bigoplus$ (a generic operator) is to be substituted with $\wedge$ if the activity has the and_seq, and_par, ord, or sync decomposition operator; and $\vee$ if the activity has the or_seq, or_par, or xor decomposition operator. Since optor_seq and optor_par enforce no restrictions, the first conjunct is eliminated when the activity has either of these decomposition operators.

The *Reset* condition is true in two situations. First, when an activity repeats (through an *Executing*-to-*Executing* transition with reset), every descendant activity or action (any activities or actions that decompose from the repeating activity) will *Reset*. Second, any top-level activity, will have its *Reset* condition automatically be true when it enters the *Done* state. When this activity *Resets*, all of its descendant activities and actions will *Reset*. More details on the EOFM formal semantics can be found in Bolton et al. (2011, 2016) and Bolton and Bass (2017).

The behavior of action outputs and local variable assignments are dependent on the action formal semantics. For *AutoReset* action behavior, the human action output is treated as Boolean (true when the action is occurring and false otherwise) and occurs when a corresponding action node in the EOFM task structure is *Executing* and does not at any other time. For an action with *toggle* behavior, the human action switches between occurring and not occurring when the corresponding action node is *Executing*. An action with *SetValue* behavior will set the corresponding human action's value when the action is *Executing*. An action with a local variable (an internal action) has its value assigned when the action is *Executing*.

3.2. An example for illustrating concepts

To facilitate the discussion of our erroneous behavior taxonomy, we are going to use a simple running example: a coffee machine (Fig. 3). In this example, we assume that the coffee machine brews coffee using water and pods that a user has entered into the machine.

The tasks for interacting with this machine are shown in Figs. 4 and 5. Fig. 4 shows the task behavior a user would use to initiate the brewing process. Fig. 5 shows the behavior for retrieving a brewed cup of coffee and properly cleaning up the machine. The task model takes inputs that indicate:

- The state of the machine's power light (iPowerOn: which is true when the light is on and false otherwise);
- The state of the machine's reservoir lid (iLidClosed: which is true when the lid is closed and false otherwise);

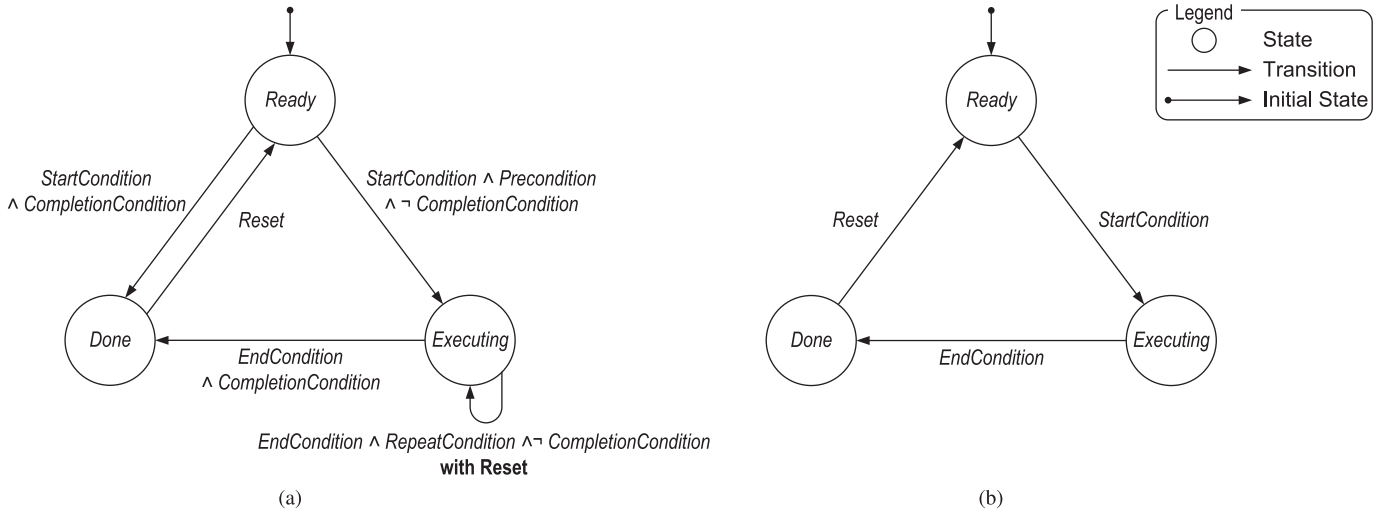

Fig. 2. Formal semantics of an EOFM (a) activity's and (b) action's execution state presented as finite state transition systems (Bolton et al., 2011). Transitions are labeled with Boolean expressions that allow a transition to occur when they are true. Note that **with Reset** is used to indicate a situation where all descendant activities and actions are *Reset* when the associated transition occurs.

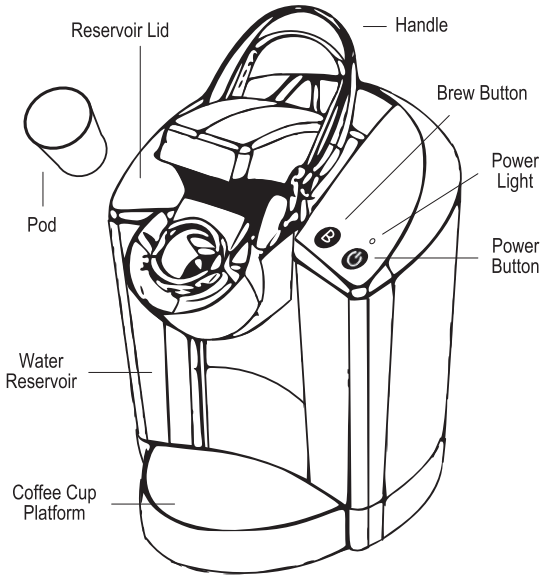

Fig. 3. A pod-based coffee machine.

- An indication of whether or not that machine has enough water (iEnoughWater: which is true if there is enough water and false otherwise);
- A variable representing the state of the machine's handle (iHandleUp: which is true when the handle is up and false when it is down);
- A variable indicating the state of the mug (iMugState; which can be Absent if no mug is placed, Empty if the mug is placed and empty; and Filled if the mug is placed and full);
- A light indicating if the machine is brewing (iBrewing: which is true if the machine is brewing and false otherwise); and
- The state of the pod in the machine (iPodState: which can have three values indicating if the pod is Fresh, Used, or Absent).

The human can, in turn, perform a number of actions on the machine. He or she can:

- Press the power button (hPressPowerButton);
- Open and close the reservoir lid (hOpenLid and hCloseLid respectively);
- Pour water into the reservoir (hPourWater);
- Lift and lower the handle (hLiftHandle and hLowerHandle respectively);
- Enter (hEnterPod) multiple types of pods (i.e. a CoffeePod or TeaPod);
- Remove a pod (hRemovePod);
- Place and remove a mug (hPlaceMug and hRemoveMug respectively); and
- Press the brew button (hPressBrewButton).

The user can perform the task for preparing for and brewing coffee (aBrewCoffee; Fig. 4) if its precondition is satisfied: that the coffee machine is not brewing and the mug is not full. This is accomplished by doing two activities in order: preparing the machine (aPrepMachine) and initiating brewing (aBrew). To prepare the machine, the user performs four sub-activities, one at a time, in any order: if the power is off, turning it on (aTurnOn) by pressing the power button (hPressPower); if there is not enough water, continually adding water (hPourWater via the aPourWater activity) until enough water has been deposited; placing the mug (hPlaceMug via aPlaceMug) if no mug is present; and inserting a pod into the machine (aAddPod). This last activity decomposes into four sub-activities that are performed in the specified order (from left to right; Fig. 4). If the handle is down, the human lifts the handle (hLiftHandle via aLiftHandle); if a used pod is in the machine, the user removes it (hRemovePod via aClearOldPod); if the pod is absent, the user inserts a fresh coffee pod (hEnterPod via aEnterPod, where hEnterPod inputs a CoffeePod); and, if the handle is up, lowering the handle (hLowerHandle via aLowerHandle). Once aPrepMachine is *Done*, the user can initiate brewing (aBrew) by pressing the brew button (hPressBrew).

The task for retrieving a brewed cup of coffee and cleaning up the machine (aGetCoffee; Fig. 5) can occur if the machine is not brewing and the mug is filled. This occurs by performing three sub-activities, one at a time, in any order: turning the power off (aPowerOff) if it is on; clearing the used pod (aClearPod) by lifting the handle, removing the pod, and then lowering the handle; and picking up the now brewed cup of coffee (aGetMug).

It is important to note the task model as presented here is a structural representation of the human behavior. The precise ways that the task can execute are determined by its formal semantics, which were discussed in the previous section. Specifically, every activity and action

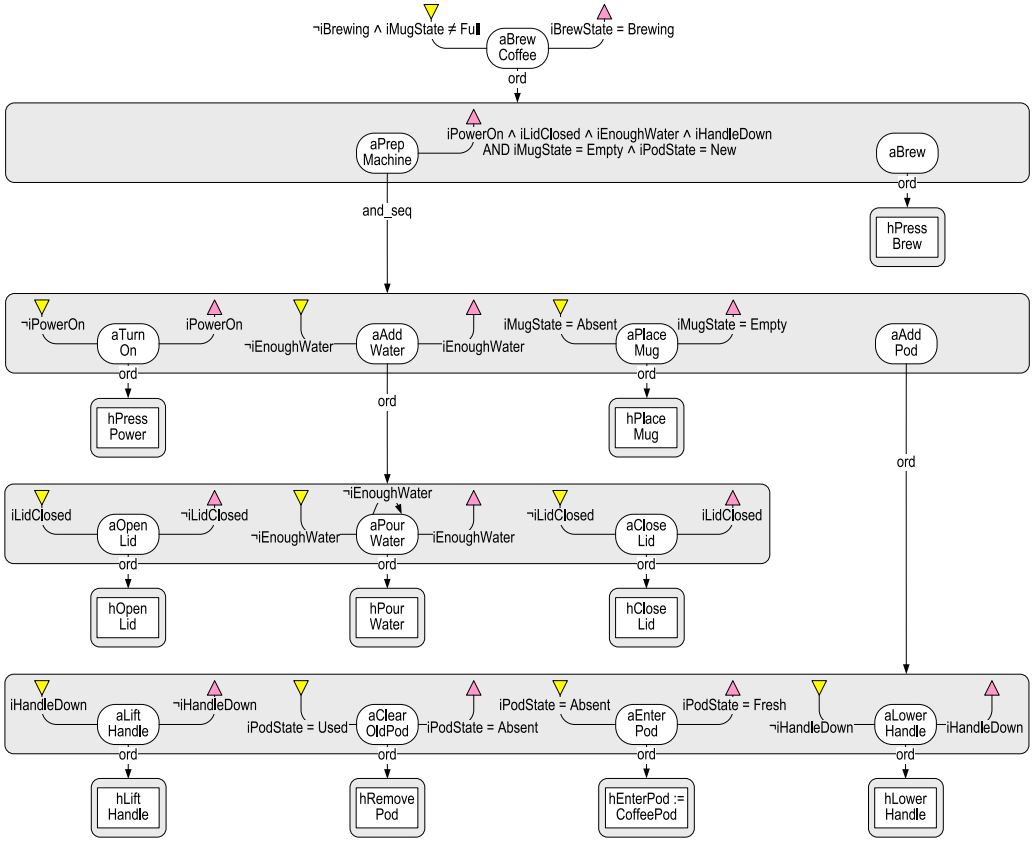

Fig. 4. EOFM task behavior model for describing how a human would prepare the machine for brewing and initiate brewing.

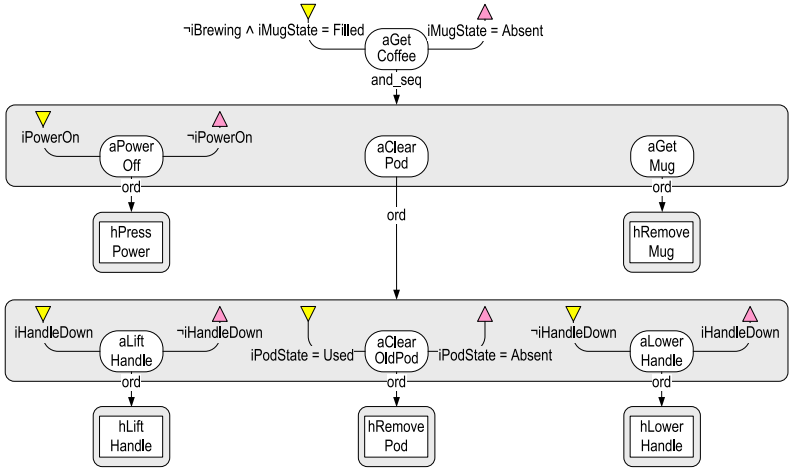

Fig. 5. EOFM task behavior for retrieving a brewed cup of coffee and performing the requisite machine maintenance.

in Figs. 4 and 5 is interpreted as a finite state machine with transitions as described in Fig. 2(a) and (b) respectively. As previously described, the *Preconditions*, *RepeatConditions*, and *CompletionConditions* come explicitly from the task model. However, *StartConditions*, *EndConditions*, and *Resets* are implicit in the model. For example, the activity aAddWater (Fig. 4) would have the *StartCondition*

$$aPrepMachin.State = Executing \wedge \begin{pmatrix} aTurnOn.State \neq Executing \\ \wedge aPlaceMug.State \neq Executing \\ \wedge aAddPod.State \neq Executing \end{pmatrix} \quad (4)$$

and the end condition

$$\begin{pmatrix} aOpenLid.State = Done \\ \wedge aPourWater.State = Done \\ \wedge aCloseLid.State = Done \end{pmatrix} \wedge \begin{pmatrix} aOpenLid.State \neq Executing \\ \wedge aPourWater.State \neq Executing \\ \wedge aCloseLid.State \neq Executing \end{pmatrix} \quad (5)$$

based on the semantics from Section 3.1 and (1) and (3).

3.3. The taxonomy

For our taxonomy, we want to classify erroneous behaviors based on where they deviate from a human operator's task and why they do

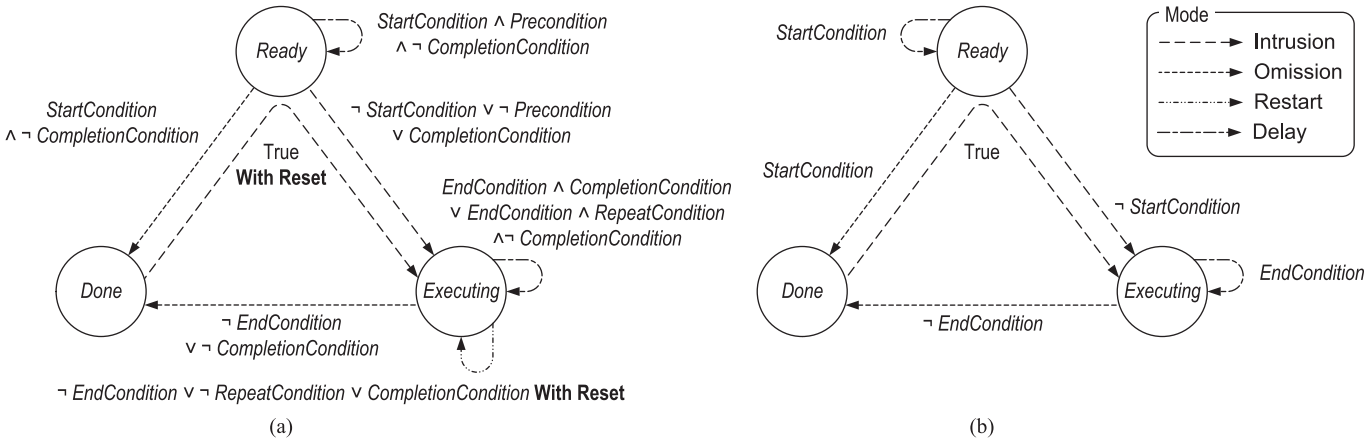

Fig. 6. Erroneous transitions in (a) activity and (b) action formal semantics. The type of dotted line used in a transition indicates the erroneous behavior mode (see the legend). All of the transitions, with the exception of the *Done*-to-*Executing*, activity *Executing*-to-*Executing* (without a *Reset*), action *Ready*-to-*Ready*, and action *Executing*-to-*Executing* ones, have guards that are the negation of the corresponding transitions from Fig. 2. The *Done*-to-*Executing* transitions represent erroneous resets where the human performs an activity or action when it should be *Done*. These are labeled with true because they do not occur in the normative semantics (Fig. 2) and thus are always erroneous. The activity *Executing*-to-*Executing* (without a *Reset*), action *Ready*-to-*Ready*, and action *Executing*-to-*Executing* transitions allow an activity or action to stay in its current state when it should transition out of it.

so. The EOFM formal semantics explicitly describe how a task should execute normatively. As such, a human who erroneously diverges from a task will do so in a way that violates the task model's formal semantics for a particular activity or action. Thus, by classifying erroneous human behavior based on which formal semantics are violated (not adhered to) and how they are violated, we will be able to identify several phenomena. First, we will know from which activity or action the divergent behavior occurred, identifying where in the task the erroneous act originated. Second, we will know what the erroneous behaviors are (what the person does) that result from the semantic violation. Finally, we will know which part of the formal semantics was violated indicating what the human operator failed to properly attend to and thus why the erroneous behavior occurred.

Our taxonomy is hierarchical (depicted later in Fig. 7) in that it classifies erroneous behavior at several levels based on the task structure and task formal semantics. First, because of differences in the formal semantics between activities (such as aBrewCoffee; Fig. 4) and actions (such as hPressPower; Fig. 4), the taxonomy distinguishes between erroneous behaviors that originate at the activity or action **level**s. Task execution can diverge from the formal semantics through violations of the execution state transitions semantics (Fig. 2) or through incorrect action variable assignments that occur during the action's execution. Thus, our taxonomy next identifies the **divergence type** associated with the erroneous act. These divergences have limited ways they can manifest, what we call the **erroneous behavior mode**s. For transition-based erroneous behavior, the human can do something (an activity or action) that they should not have done (an intrusion), not do an activity or action they should have done (an omission), restart the execution of something (a restart), or fail to perform a transition when it is supposed to (a delay). For execution-based erroneous behaviors, a person can replace a value or variable with a correct one or misremember something. These modes represent the next level in the taxonomy. Within each of these modes, we further classify an erroneous behavior based on the specific **point of divergence** from the semantics. For transition-based erroneous behaviors, this represents the specific type of erroneous transition that occurred. For execution-based erroneous behaviors, this is the type of variable assignment being used (the type of action being performed). Finally, points of divergence are further refined and given meaningful **specific erroneous behavior types/names** based on the information/condition the human improperly attended to (for transition-based erroneous behaviors) or improper variable assignment (for execution-based errors). A full representation of the hierarchy is shown later in Fig. 7.

The following sections further elaborate on the details of the taxonomy. First, we discuss the transition-based erroneous behaviors at both the activity and action-levels. This is followed by a discussion of the execution-based erroneous behaviors.

3.3.1. Transition-based erroneous behaviors

We classify transition-based erroneous behaviors into modes based on the type of behavior that can result from an erroneous transition. The human can do something (an activity or action) that is not his or her current activity or action (an intrusion), fail to do the correct activity or action (an omission), restart an already executing activity (a restart), or fail to transition when it is supposed to (a delay).

Fig. 6 shows the formal state transition semantic violations we include in the taxonomy and their associated erroneous behavior mode: intrusion, omission, restart, or delay. The formal semantics represented here do not encompass all of the possible violations that can occur. Some possible transitions were excluded because they are either artificial constructs that make sense for the finite automata perspective, but not from a human error perspective, or they are encapsulated by other transitions. The discussion below describes each of the included transitions as well as the transitions that were excluded.

The guards on the *Ready*-to-*Done*, *Ready*-to-*Executing*, *Executing*-to-*Done*, and activity *Executing*-to-*Executing* (with reset) transitions each represent the negation of the guards on the corresponding conditions in Fig. 2. That is, they occur in any condition outside of the normative transitions. Erroneous *Done*-to-*Ready* transitions are not included because they would only result in erroneous behavior if the associated activity or action then transitioned to *Executing*. Thus, the *Done*-to-*Executing* transitions in Fig. 6 represent the erroneous analogs to the *Reset* conditions in the normative semantics (Fig. 2). The guard on this condition is true because any such transition would always be erroneous. The *Ready*-to-*Ready* and *Executing*-to-*Executing* (Fig. 6(a)) transitions (delays) occur in situations where there should be transitions from *Ready*-to-*Executing* and *Executing*-to-*Done* (respectively). Note that the delay *Executing*-to-*Executing* transitions do not issue a reset to the associated activity's descendants, thus differentiating them from restart transitions. Further note that a comparable *Done*-to-*Done* transition is not included in the erroneous transitions. This is discussed further below.

The erroneous behavior modes and their associated erroneous transitions (points of divergence) are useful for identifying how an erroneous behavior manifests and where the divergence occurs. However, they do not provide insights into exactly why an erroneous behavior occurred (what the human inappropriately attended to that caused the erroneous act). To address this, we can further refine our classification based on which conditions in the guard were violated. This determines the specific erroneous behavior type. Table 3 shows all of the specific erroneous

Table 3
Activity-level, transition-based erroneous behaviors.

Mode	Transition / Point of divergence	Transition condition	Erroneous behavior type
Intrusion	*Ready*-to-*Executing*		
		$\neg StartCondition \wedge Precondition \wedge \neg CompletionCondition$	Activity Capture Intrusion
		$StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge \neg CompletionCondition$	Activity Premature Intrusion
		$StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge \neg CompletionCondition$	Activity Premature, Repeat Intusion
		$StartCondition \wedge Precondition \wedge CompletionCondition$	Activity Completed Intrusion
		$\neg StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge \neg CompletionCondition$	Activity Spurious Intrusion
		$\neg StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge \neg CompletionCondition$	Activity Repeat-Capture Intrusion
		$\neg StartCondition \wedge Precondition \wedge CompletionCondition$	Activity Completed, Capture Intrusion
		$StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge CompletionCondition$	Activity Completed, Premature Intrusion
		$StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge CompletionCondition$	Activity Completed, Premature, Repeat Intusion
		$\neg StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge CompletionCondition$	Activity Completed, Spurious Intrusion
		$\neg StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge CompletionCondition$	Activity Completed, Repeat-Capture Intrusion
	Done-to-*Executing*		
		$StartCondition \wedge Precondition \wedge \neg CompletionCondition$	Activity Reset Intrusion
		$\neg StartCondition \wedge Precondition \wedge \neg CompletionCondition$	Activity Capture, Reset Intrusion
		$StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge \neg CompletionCondition$	Activity Premature, Reset Intrusion
		$StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge \neg CompletionCondition$	Activity Premature, Repeat, Reset, Intusion
		$StartCondition \wedge Precondition \wedge CompletionCondition$	Activity Completed Intrusion
		$\neg StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge \neg CompletionCondition$	Activity Spurious, Reset Intrusion
		$\neg StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge \neg CompletionCondition$	Activity Repeat-Capture, Reset Intrusion
		$\neg StartCondition \wedge Precondition \wedge CompletionCondition$	Activity Completed, Capture, Reset Intrusion
		$StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge CompletionCondition$	Activity Completed, Premature, Reset Intrusion
		$StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge CompletionCondition$	Activity Completed, Premature, Repeat, Reset Intusion
		$\neg StartCondition \wedge \neg Precondition \wedge \neg RepeatCondition \wedge CompletionCondition$	Activity Completed, Spurious, Reset Intrusion
		$\neg StartCondition \wedge \neg Precondition \wedge RepeatCondition \wedge CompletionCondition$	Activity Completed, Repeat-Capture, Reset Intrusion
Omission	*Ready*-to-*Done*		
		$StartCondition \wedge \neg CompletionCondition$	Activity Omission
	Executing-to-*Done*		
		$\neg EndCondition \wedge CompletionCondition$	Activity Post-Completion Omission
		$EndCondition \wedge \neg CompletionCondition$	Activity Non-completion Omission
		$\neg EndCondition \wedge \neg CompletionCondition$	Activity Spurious Termination Omission
Restart	*Executing*-to-*Executing* with Reset		
		$\neg EndCondition \wedge RepeatCondition \wedge \neg CompletionCondition$	Activity Post-Repetition
		$EndCondition \wedge \neg RepeatCondition \wedge \neg Precondition \wedge \neg CompletionCondition$	Activity Premature Restart
		$EndCondition \wedge \neg RepeatCondition \wedge Precondition \wedge \neg CompletionCondition$	Activity Premature, Pre-capture Restart
		$EndCondition \wedge RepeatCondition \wedge CompletionCondition$	Activity Completed Restart
		$\neg EndCondition \wedge \neg RepeatCondition \wedge \neg Precondition \wedge \neg CompletionCondition$	Activity Spurious Restart
		$\neg EndCondition \wedge \neg RepeatCondition \wedge Precondition \wedge \neg CompletionCondition$	Activity Spurious, Pre-capture Restart
		$\neg EndCondition \wedge RepeatCondition \wedge CompletionCondition$	Activity Completed, Post-Repetition
		$EndCondition \wedge \neg RepeatCondition \wedge \neg Precondition \wedge CompletionCondition$	Activity Completed, Premature Restart
		$EndCondition \wedge \neg RepeatCondition \wedge Precondition \wedge CompletionCondition$	Activity Completed, Premature Pre-capture Restart
		$\neg EndCondition \wedge \neg RepeatCondition \wedge \neg Precondition \wedge CompletionCondition$	Activity Completed, Spurious Restart
		$\neg EndCondition \wedge \neg RepeatCondition \wedge Precondition \wedge CompletionCondition$	Activity Completed, Spurious, Pre-capture Restart
Delay	*Ready*-to-*Ready*		
		$StartCondition \wedge Precondition \wedge \neg CompletionCondition$	Activity Start Delay
	Executing-to-*Executing*		
		$EndCondition \wedge CompletionCondition$	Activity Finish Delay
		$EndCondition \wedge RepeatCondition \wedge \neg CompletionCondition$	Activity Repeat Delay

Note. Each erroneous behavior type's transition condition satisfies the associated erroneous transition from Fig. 6(a). Underlines are used to show where in a transition condition a deviation from normative occurs. For the *Done*-to-*Executing* transitions, there is an assumed erroneous reset transition. Note that any given activity may not have all of the strategic knowledge conditions. If an activity does not explicitly define a particular strategic knowledge condition, there is an assumed default value, where the default values depend on the transition. For erroneous *Ready*-to-*Done*, *Ready*-to-*Executing*, and *Done*-to-*Executing*: a *Precondition*'s default value is equal to the *StartCondition* and a *RepeatCondition* and a *CompletionCondition* are false by default. For erroneous *Executing*-to-*Executing* and *Executing*-to-*Done* transitions: a *Precondition* and *RepeatCondition* are false by default and a *CompletionCondition* is, by default, equal to the *EndCondition*.

behavior types for activity-level erroneous behaviors. Each entry in this table shows the exact condition (listed under transition condition) under which a given erroneous transition / point of divergence occurs. Underlines are used to show which conditions are violated in the transition (which condition are improperly attended to by the human). Erroneous transitions are also given a descriptive name (erroneous behavior type) to relate the encapsulated concepts to ideas familiar to human factors engineers. These names are convenient for users of the taxonomy because they account for the conveyed levels, transitions, and condition violations without requiring potential users to explicitly refer to the underlying erroneous semantics. Below we discuss each of the different erroneous transitions and their associated specific erroneous behavior types.

Activity Ready-to-Executing Intrusions. For activity-level *Ready*-to-*Executing* intrusions, there are 11 erroneous behaviors. An activity capture intrusion (a classic capture error) occurs when the strategic knowledge conditions for an activity are satisfied, but it is not the

appropriate time to perform that activity in the task (the *StartCondition* is not true). Thus, an activity capture intrusion occurs when the human fails to attend to the *StartCondition* properly. An example of this could occur with the coffee machine when the human is performing the task for getting the coffee (Fig. 5) when the mug is placed and the machine is no longer brewing. In this situation, if there is not enough water in the reservoir, the human could erroneously perform the activity for adding water to the machine (aAddWater; Fig. 4) because its *Precondition* (¬ *iEnoughWater*) is satisfied but its *StartCondition* is not ($aPrepMachin.State = Executing$ is false; see (4)).

An activity premature intrusion occurs when it would otherwise be appropriate for the activity to start executing (its *StartCondition* is true), but without the *Precondition* or *RepeatCondition* being satisfied. Thus, it occurs when the human fails to properly attend to these condition. Note that *RepeatConditions* only show up in transition conditions of *Ready*-to-*Executing* intrusions (Table 3) when the *Precondition* is not satisfied. This is because it is irrelevant when the *Precondition* is true. For the coffee machine, this could occur after the human has completed the task for brewing the coffee (the task in Fig. 4) and the machine is brewing (iBrew is true) with the mug in place (iMugPlaced is also true). In this situation, aGetCoffee's *StartCondition* is true because aBrewCoffee is not executing. However, its *Precondition* is not true because ¬*iBrew* is false. Thus, a premature intrusion occurs if somebody attempts to execute aGetCoffee.

An activity completed intrusion occurs when both the *StartCondition* and *Precondition* are properly satisfied (if there is indeed a precondition on the activity), but when the *CompletionCondition* has been satisfied. Thus, the activity does not need to execute because its goal (*CompletionCondition*) has been achieved. For the coffee application, this could happen if the human is performing the task for brewing the coffee (aBrewCoffee; Fig. 4) but with the machine already configured for brewing: the power is on, the lid is closed, there is enough water, the handle is down, the mug is placed and empty, and there is a fresh pod. The human can make an activity completed intrusion if he or she performs the aPrepMachine activity when its *StartCondition* ($aBrewCoffee.State = Executing \wedge aBrew.State \neq Executing$) and *Precondition* are true, even though its *CompletionCondition* is satisfied.

An activity repeat-capture intrusion is just like an activity capture intrusion except that it occurs when a human's attention is captured by the *RepeatCondition* instead of the *Precondition*. There are no good examples of this in the coffee machine task.

The remainder of the *Ready*-to-*Executing* intrusions are variations/combinations of these erroneous behaviors.

Activity Done-to-Executing Intrusions. There are 12 activity *Done*-to-*Executing* intrusions, all dubbed reset intrusions. The base activity reset intrusion occurs under the condition where an activity would transition for *Ready*-to-*Executing* normatively. The remainder of the *Ready*-to-*Executing* intrusions are the analogues of *Ready*-to-*Executing* intrusions, only differing in that they occur from an activity's *Done* state. Consider the example in the coffee machine application where the human is performing aGetCoffee (Fig. 5). In this, assume that he or she first properly performs the activity for clearing the pod (aClearPod) and then performs one or more of the other activities in the decomposition (aPowerOff and/or aGetMug). An activity reset intrusion will occur if the human restarts the performance of aClearPod (thus resetting all of its decomposed activities and actions) as if the activity had not previously been done.

Activity Omissions. There are four activity omissions. One for *Ready*-to-*Done* and three for *Executing*-to-*Done* transitions.

The *Ready*-to-*Done* omission occurs when it is appropriate to do the activity (the *StartCondition* is true) and the goal of the activity has not been satisfied (the *CompletionCondition* is false), but the activity is not performed because the human treats the completion conditions as if it is true. For the coffee machine, this could happen when the human is performing aBrewCoffee (Fig. 4) and does not properly attend to the state of the machine as prescribed in aPrepMachine's completion condition. This could result in the person thinking aPrepMachine is completed, not performing the activity, and moving on to aBrew without properly preparing the machine.

For *Executing*-to-*Done*, an activity post-completion omission occurs when its *CompletionCondition* is satisfied, but its *EndCondition* is not. Note that this is a classic post-completion error (Byrne and Bovair, 1997; Li et al., 2005), a type of erroneous behavior where a human fails to complete sub-goals/activities in a task because the primary goal has been achieved. The best example of this in the coffee machine task would occur under aGetCoffee (Fig. 5). If the human operator retrieves the filled coffee mug (aGetMug) before performing the other two activities (aPowerOff and aClearPod), the human may erroneously stop performing aGetCoffee because its completion condition (iMugState = Absent) is satisfied even though its *EndCondition* is not (aPowerOff and aClearPod are not *Done*).

The opposite of an activity post-completion omission is an activity non-completion omission. This occurs when the activity's *EndCondition* is satisfied, but its goal (*CompletionCondition*) has not been achieved. In the coffee machine, this is best illustrated by aPourWater under aBrewCoffee (Fig. 4). If the human is performing aPourWater and has just completed hPourWater, but without yet adding enough water for iEnoughWater to become true, aPourWater's *EndCondition* is true, but not its *CompletionCondition*. Thus, if the human treats aPourWater as if it is *Done* and moves on to aCloseLid, then an activity non-completion omission has occurred.

An activity spurious termination omission is a combination of the previous two erroneous behaviors in that the omission occurs in violations of an activity's *EndCondition* and *CompletionCondition*. In the coffee machine, this could occur if the human stops performing aPrepMachine (under aBrewCoffee; Fig. 4) before completing all of its sub-activities (violating its *EndCondition*) and before its completion condition is satisfied.

Activity Restarts. Activity restarts, of which there are 11, all occur due to erroneous *Executing*-to-*Executing* transitions, where a reset is broadcast to all descending activities and/or actions when the transition occurs. These are conceptually similar to the *Ready*-to-*Executing* intrusions except that the role of the *RepeatConditions* and *Preconditions* are reversed. Note that an activity post-repetition occurs when a *RepeatCondition* is true, but the activity has not yet reached its *EndCondition*. Also note that an erroneous act with "pre-capture" in its name indicates an erroneous behavior where a *Precondition* captures human attention in an inappropriate situation. All of the other erroneous restarts should be relatively straightforward to interpret.

It is important to note that activity restarts can occur when an activity has a repeat condition and when it does not. For example, for the coffee machine example, aPourWater (which has a repeat condition; see Fig. 4) could be performed with an activity completed, premature restart, if the human repeats that activity even though enough water has been added to the reservoir. Alternatively, an activity premature restart could occur for aBrewCoffee (Fig. 4; which does not have a *RepeatCondition*) if the human erroneously restarts the activity any time during the execution of its sub-activities.

It is important to note that a previously executed activity can execute again. However, in our taxonomy, this will occur via activity done-to-executing intrusions, not activity restarts.

Activity Delays. Activities can also have delays. A *Ready*-to-*Ready* transition that occurs when an activity should normatively transition from *Ready*-to-*Executing* represents an activity start delay. *Executing*-to-*Executing* transitions (which do not broadcast a reset) can result in two different types of delays.

Table 4
Action-level erroneous behaviors.

Divergence type	Mode	Point of divergence	Transition condition	Erroneous behavior type
Transition	Intrusion	*Ready*-to-*Executing*	¬*StartCondition*	Action Intrusion
		Done-to-*Executing*	*StartCondition*	Action Reset Intrusion
			¬*StartCondition*	Action Spurious Reset Intrusion
	Omission	*Ready*-to-*Done*	*StartCondition*	Action Omission
		Executing-to-*Done*	¬*EndCondition*	Action Premature Finish
	Delay	*Ready*-to-*Ready*	*StartCondition*	Action Start Delay
		Executing-to-*Executing*	*EndCondition*	Action Finish Delay
Divergence type	**Mode**	**Point of divergence**	**Assignment**	**Erroneous behavior type**
Execution	Substitution	SetValue	*CorrectAction* := *IncorrectValue*	Action Value Substitution
			IncorrectAction := *CorrectValue*	Action Target Substitution
	Misremembrance	Local Variable Assignment	*CorrectVariable* := *IncorrectValue*	Value Misremembrance
			IncorrectVariable := *CorrectValue*	Target Misremembrance

Note. Each erroneous behavior type in the top part of the table satisfies the associated erroneous transitions in Fig. 6(a). Underlines are used to show where in a transition condition or a variable assignment a deviation from normative occurs.

If the human does not complete the activity when he or she should (when the conditions for an *Executing*-to-*Done* transition are true), an activity finish delay occurs. Such a condition could occur in a number of activities in our coffee machine application. For example, if the human pauses after lifting the handle of the machine (aLiftHandle; in either Fig. 4 or Fig. 5), this would constitute an activity finish delay.

If the human does not perform a true repetition (an *Executing*-to-*Executing* transition with a reset), an activity repeat delay occurs. In the coffee machine, this could occur if the person stops or pauses while adding water to the reservoir (aPourWater; Fig. 4) before enough water has been added.

Note that a comparable delay for a *Done* activity (an erroneous *Done*-to-*Done* transition) is not included in our taxonomy. This is because such a transition would be an artifact of the finite state machine representation: a human does not think about an activity or action being reset (the only way to transition out of *Done*) and thus a delay on a reset transition is artificial. Further, the only way an erroneous *Done*-to-*Done* transition would impact the performance of human behavior is if a human fails to perform the given activity or action after its parent activity has been reset and is executing. Such behavior would occur because the person is not properly attending to the conditions indicating when that activity or action should be performed and is thus encapsulated by the erroneous *Ready*-to-*Done* transitions.

Action-Level Transition-Based Erroneous Behaviors. Erroneous transition-based behavior can also occur at the action level. As with the activity transitions, these can be refined based on the property that is violated. However, because actions do not have strategic knowledge conditions, there are fewer transition-based erroneous behaviors at this level. These are shown in the upper half of Table 4.

An action intrusion occurs when an action transitions from *Ready* to *Executing* at the wrong time (when the human fails to properly attend to the *StartCondition*). A *Done*-to-*Executing* action reset intrusion occurs when the person performs an action that is *Done*, but has its *StartCondition* satisfied. An action spurious intrusion describes a condition where this occurs when the action's *StartCondition* is not satisfied. An action omission occurs when an action transitions from *Ready* to *Done* when it is the correct time to transition from *Ready* to *Executing* (the *StartCondition* is true). An action premature finish (a special type of action omission) occurs when an action transitions from *Executing* to *Done* before the action's *EndCondition* is true. An action start delay occurs when the activity transitions from *Ready*-to-*Ready* when it normatively should have transitioned from *Ready*-to-*Executing*. An action finish delay occurs when the activity transitions from *Executing*-to-*Executing* when it normatively should have transitioned from *Executing*-to-*Done*. Every action in our coffee machine application could be used to illustrate these erroneous behaviors.

The action-level (Table 4) behaviors do not explicitly account for repetition. This is because, for an action to finish and then repeat, it needs to transition to *Done*. A direct *Executing*-to-*Executing* transition would thus not register the completion of the action. As such, action repetition is encapsulated by an action reset intrusion, where this occurs after the action transitions from *Executing* to *Done*.

Additional Transition-based Considerations. It is important to note that each of the transition-based erroneous behavior types can be further refined based on which part of the violated condition was not attended to. For example, a human may perform an action intrusion, where the elements in the *StartCondition* that are violated are related to the execution state of the action's siblings (the other actions in the decomposition) rather that the execution state of the parent (the activity the actions decomposes from). In this situation, erroneous behavior will result in actions being executed in the wrong order or instead of another action, depending on the activity's decomposition operator. Similar distinctions can be made for all other errors and condition types. This is further discussed in Section 4.

3.3.2. Execution-based erroneous behaviors

The transition-based, action-level erroneous behaviors can account for people doing the wrong action or omitting the correct action. However, when an action involves the conveyance of information beyond simple performance (as is done for *SetValue* actions and local variable assignments), there are additional deviations that can occur in the assignment process. These are shown in the bottom of Table 4.

These execution-based erroneous behaviors are separated into two modes. A substitution is associated with *SetValue* actions (the point of divergence) and represents part of an action's value conveyance being substituted with something correct. A misremembrance is associated with local variable assignments (the point of divergence). It models a person remembering something partially wrong when performing a mental action. The erroneous behavior types within each of these designations are distinguished based on whether the target (the action or local variable) is incorrect or the value assigned to it is incorrect. Note that a condition where both are wrong is still possible in the larger taxonomy. This can occur either through an intrusion or through the combination of an intrusion and an action omission.

For substitutions, the human can perform an action where they convey the wrong value through the correct action (an action value substitution). In the coffee machine example, this could occur by the human operator placing something other than a coffee pod into the machine (performing an action value substitution for hEnterPod under aEnterPod in Fig. 4). If the person performs the wrong action, but the right value, this is called an action target substitution. For the coffee machine, for the same place in the task, this could occur if the human operator placed the coffee pod somewhere other than in the machine.

Analogues for these can also occur for misremembrances (local variable assignments). Someone can remember the wrong thing (a value misremembrance) or remember the right thing in the wrong context (a target misremembrance). However, because there are no local variable assignments in the coffee machine tasks, there are not illustrative examples of misremembrances for the application.

3.4. Summary

Fig. 7 provides an overview of the hierarchy used in our taxonomy. It is worth noting that although each of these categories is general, any given erroneous behavior that actually occurs will be associated with a specific activity or action. Thus, additional context and insights can be had by considering the behaviors being performed and any strategic knowledge that factored into the erroneous behavior.

4. Inter-taxonomy compatibility

Our new taxonomy is intended to encompass both phenomenological and genotypical erroneous behavior concepts. To assess this, we show that our taxonomy achieves coverage over both Hollnagel's phenotypes of erroneous action (1993) and slips from Reason's GEMS (1990). We do this by going through all of the designations of the other taxonomies and showing how each of their classifications is accounted for in the new system.

4.1. Phenotypes of erroneous action

Hollnagel's phenotypes of erroneous action (1993) were concerned with how erroneous behaviors manifest as deviations from a normative plan or task. Thus, the discussion below describes how each of Hollnagel's phenotypes can manifest using our taxonomy.

4.1.1. Zero-order phenotypes

Premature Start. A premature start occurs when an action starts too early (Hollnagel, 1993). In our taxonomy, premature starts can occur in several ways. At the action level, a premature start can occur when a person performs an action intrusion right before the action's *StartCondition* is true. However, a premature start can also occur for similar reasons at the activity level if a human performs an activity capture intrusion right before the activity's *StartCondition* is satisfied. This can result in one or more of the actions that the activity decomposes into executing prematurely. Alternatively, timing constraints can potentially be asserted into an activity's *Precondition* or *RepeatCondition*. If this is the case and a human does not properly attend to these elements of an activity's *Precondition* or *RepeatCondition*, he or she may perform an activity premature intrusion; an activity premature, repeat intrusion; an activity premature restart; or an activity premature pre-capture restart.

Delayed Start. A delayed start describes occurs where an action does not start when it is supposed to (Hollnagel, 1993). In the new taxonomy, this can occur when the human fails to start the execution of an action due to improperly attending to the *StartCondition*, an action start delay. Delays at the activity level can also occur. A human may fail to attend to the conditions when an activity should normatively transition from *Ready*-to-*Executing* and perform an activity start delay. A human may also fail to attend to the conditions associated with when the activity should normatively repeat and perform an activity repeat delay.

Premature Finish. A premature finish occurs when an action finishes before it should (Hollnagel, 1993). In our taxonomy, this manifests as an action premature finish. An action can also finish prematurely if its parent activity finishes before it should. Thus, premature finishes can happen if any of the activity *Executing*-to-*Done* omissions occur while an action is executing.

Delayed Completion. A delayed completion occurs when an action does not finish when it is supposed to (Hollnagel, 1993). This is replicated by an action finish delay, where the human does not stop executing when the *EndCondition* is satisfied.

Omission. An omission occurs when the human does not perform a planned action (Hollnagel, 1993). This is replicated directly by an action omission in the new taxonomy. Action omissions can also occur if any activity-level omission occurs before the activity has executed all of its required actions.

Jump Forward. A jump forward involves a human performing an action that occurs later in a plan (Hollnagel, 1993). At the action level, this can be represented by an action intrusion, as long as the action is one that occurs later in the given activity or task. A jump can also occur for an activity via any *Ready*-to-*Executing* intrusion as long as the activity is in the same task as the normative action/activity.

Jump Backward. A jump backward occurs when a human performs a previously completed action (Hollnagel, 1993). In our taxonomy, a *Done* action or activity is one that was previously performed. Thus, a jump backward can be represented by an action reset intrusion or an action spurious reset intrusion. Similarly, a jump backward can occur for any *Done*-to-*Executing* activity intrusion, as long as the resulting actions were performed in the previous execution of the activity.

Repetition. A repetition occurs when a human repeats the action he or she just performed (Hollnagel, 1993). In the new taxonomy, this occurs when either an action reset intrusion or an action spurious reset intrusion occurs right after the completion (*Executing*-to-*Done*) of that same action. A repetition can also happen if an activity only has one action in its decomposition and any activity restart (erroneous *Executing*-to-*Executing* transitions with reset) happens after the completions of the activity's action or if the just finished activity erroneously transitions from *Done*-to-*Executing*.

Intrusion. An intrusion occurs when a human performs an unplanned action (Hollnagel, 1993). In our taxonomy, this can occur via an action intrusion, where the action is not in the task currently being performed. At the activity level, action intrusions can occur for any *Ready*-to-*Executing* intrusion where the activity that is intruded is not part of the current executing task. It is important to note that this will inherently exclude any intrusion where the *StartCondition* is satisfied (a *StartCondition* can only become true for an executing activity). It also excludes any *Done*-to-*Executing* intrusion because there would be no *Done* activities or actions in non-executing tasks.

4.1.2. First-order phenotypes

Spurious Intrusion. A spurious intrusion is represented by multiple zero-order intrusions in sequence (Hollnagel, 1993). In the new taxonomy, this can occur via a sequence of action intrusions, reset intrusions, or spurious reset intrusions. Any activity intrusions cannot be rightly qualified under this designation because it would constitute the performance of actions within a given activity that have a specific relationship with each other.

Jump / Skip. A jump / skip occurs when a human skips multiple actions (Hollnagel, 1993). This is represented in our taxonomy as a sequence of action omissions or an activity-level omission.

Place Losing. Place losing occurs when a human performs actions in an arbitrary order (Hollnagel, 1993). In the new taxonomy, information about activity and action execution order is encoded into its *StartCondition*. Thus, place losing behavior is replicated by multiple action intrusions or activity capture intrusions during the execution of a parent activity that encapsulates the activities or actions.

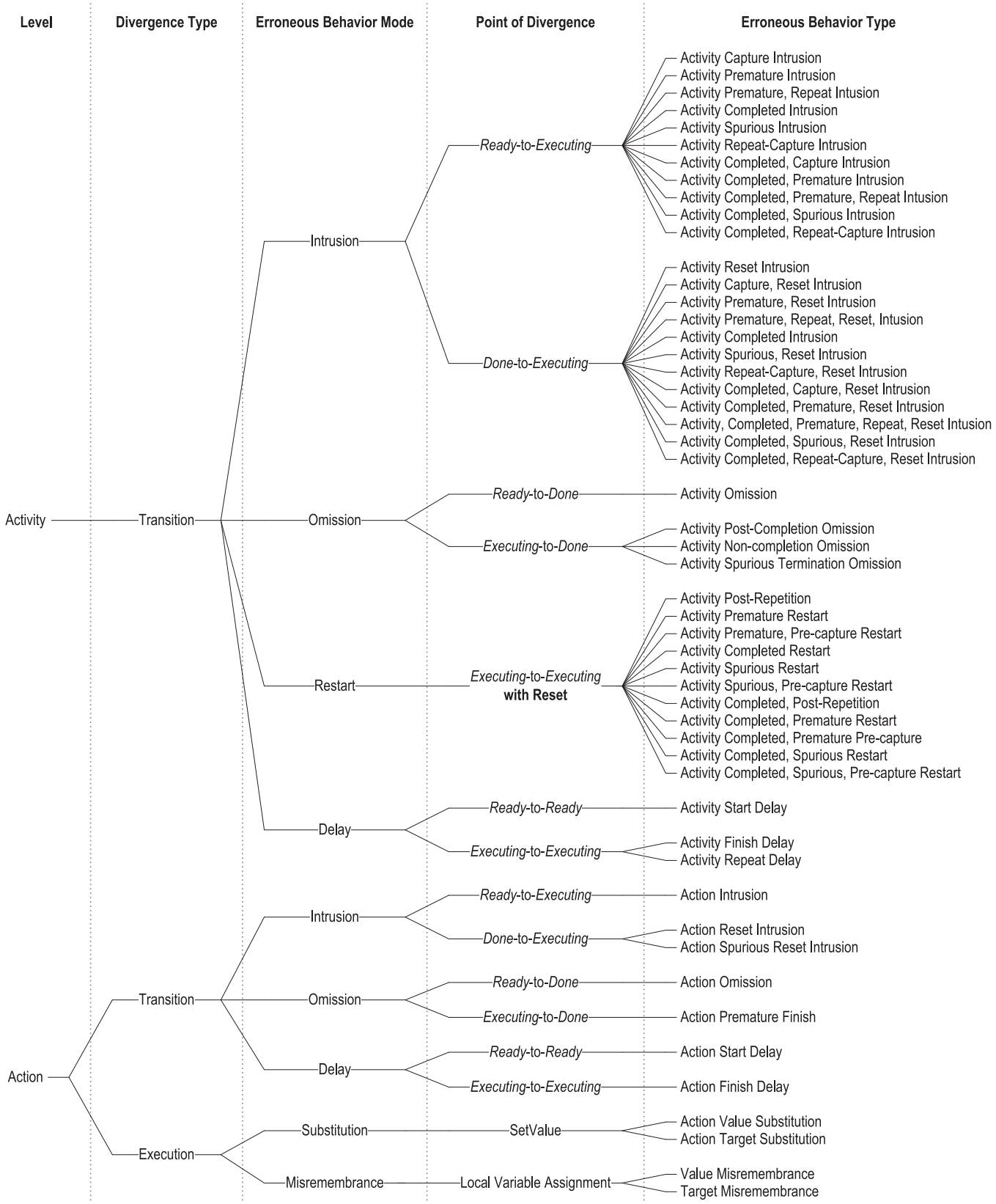

Fig. 7. Summary of erroneous behavior classifications using the new taxonomy.

Recovery. Recovery occurs when a human performs actions that were previously omitted (Hollnagel, 1993). In the new taxonomy, the fact that acts were previously omitted suggests that they will be in the *Done* state. Thus, recovery occurs when any activity or action that was previously omitted will transition from *Done* to *Executing* (any of the reset intrusions).

Side Tracking. Side tracking represents a situation when one part of an action plan is replaced with another (Hollnagel, 1993). In our taxonomy, this would require the performance of one or more omissions (at either the action or activity level) and then an activity-level intrusion, where the intruding activity would be from another task.

Capture. A capture occurs when a human performs part of another action sequence in the wrong place though multiple intrusions (Hollnagel, 1993). In our new taxonomy, this can be replicated by any activity-level intrusion. Hollnagel also discussed a special type of capture called branching, where the intruding action sequence would have started under similar conditions to the normative sequence (Hollnagel, 1993). In the new taxonomy, this corresponds with the activity intrusions that are given "capture" designations: activity repeat-capture intrusion; activity completed, capture intrusion; activity completed, repeat-capture intrusion; activity repeat-capture, reset intrusion; activity completed, capture, reset intrusion; and activity completed, repeat-capture, reset intrusion. These are appropriate for "branching" because they represent conditions where activity *Preconditions* or *RepeatConditions* indicate that the erroneous activity should be performed.

Reversal. A reversal represents a situation where the execution of two adjacent actions is reversed (a combination of an omission and a jump; Hollnagel, 1993). In our taxonomy, this would be replicated by an action intrusion, where the intruding action is one that would have executed next. Note that because an action that has executed will be *Done* when the activity would have normatively reached it. Thus, there is no need for an additional erroneous behavior.

Time Compression. Time compression occurs when a sequence of actions occurs faster than they should through multiple premature starts, premature finishes, and/or performing actions in parallel (Hollnagel, 1993). In the new taxonomy, time compression would occur in a similar manner: through multiple action intrusions (where an action starts before the actions it should wait for are *Done*) and/or action premature finishes. It is important to note that an action intrusion allows for the erroneous parallel execution of actions given that restrictions on such behavior are encoded into action *StartConditions*.

4.1.3. Discussion

The above discussion shows that our taxonomy is able to account for all of the phenotypes of erroneous action identified by Hollnagel (1993). Our taxonomy thus covers the various phenotypes found in Hollnagel's taxonomy and encompasses the contained concepts.

For the vast majority of the phenotypes, there were multiple ways that the included behavior could be represented in our taxonomy. This speaks to the capabilities of our method. Specifically, our taxonomy not only accounts for how the erroneous behavior manifested, but also why it occurred. Conversely, several first order phenotypes can only be replicated in our taxonomy through multiple erroneous behavior types. While this does not allow for elegant classifications of some of the first-order types in our taxonomy, this is not a major limitation. Specifically, our taxonomy is still able to replicate the observable behavior encapsulated by phenotypes. It is just that wedding the phenotypes to the reasons the violations occurred can require multiple error types.

Several specific erroneous behaviors from our new taxonomy are not addressed by Hollnagel's phenotypes. In particular, none of the execution-based erroneous behaviors are included. This is not surprising for misremembrances because such behaviors are not observable. The lack of substitution in Hollangel's phenotypes is a potentially more serious issue. However, a convincing argument could be made that these substitutions would be encapsulated by a zero-order intrusion. Ultimately, the fact that the new taxonomy identifies this specific distinction shows its enhanced specificity.

4.2. GEMS slips

Slips in GEMS (Reason, 1990) all represent erroneous behavior that occur when the human knows what they should be doing, but fail to do so because of a failure of attention. We discuss the relationship of our taxonomy to Reason's slips based on their association with inattention and over attention below.

4.2.1. Inattention

Double Capture Slip. A double capture slip occurs when a human's attention is captured by something else in the environment resulting in the person doing something different than what they were currently doing (Reason, 1990). Our taxonomy encompasses this behavior with the various forms of activity "capture" intrusions. Note that all of these meet with Reason's definition of a double capture slip because the *Precondition* or the *RepeatCondition* capture the human's attention.

Omission Following an Interruption. An omission following an interruption can occur as a result of a person not attending to the task following an external event (Reason, 1990). In the new taxonomy, this can occur for any type of omission. However, it best matches activity level omissions because it suggests that the distraction caused the person to fail to attend to the state of the task (encompassed by the activity's *EndCondition*) or the completeness of its goals (expressed in the *CompletionCondition*). It is important to note that omissions in the new taxonomy do not explicitly account for the interruption. Rather, they enumerate what information was not properly attended to. In the context of cognitive explanations, this is actually more utility than the presence of an external interruption.

Reduced Intentionality. Reduced intentionality takes place when the human loses track of his or her intentions in the middle of a task (Reason, 1990). In our taxonomy, this is represented by a situation where an activity erroneously transitions from *Executing*-to-*Done*, omitting subactivities or actions that should have been performed.

Perceptual Confusion. Perceptual confusion occurs in tasks where somebody performs the correct action on the wrong target due to similarities between the wrong target and the correct target (Reason, 1990). These are directly represented by action value substitutions in the new taxonomy. In these, the person performs the right action with the wrong value (an action value substitution).

Interference Errors. Interference errors happen when the human is performing two or more tasks and inappropriately blends them together. As with perceptual confusion, this is represented by execution-based substitutions and misremembrances. In this situation, the incorrect values, actions, or local variables would be values, actions, or variables from the other task, respectively.

4.2.2. Over attention

According to Reason (1990), slips also occur because of mistimed attentional checks due to over attention. This can cause a human to omit something (an omission), repeat something that was already done (a repetition), or reverse the order of steps in a sequence (a reversal). Over attention can cause a human to incorrectly evaluate and/or ignore any of the conditions from a task model.

An omission is accounted for in our taxonomy through any of the activity or action-level omissions.

Repetitions are included in our taxonomy via activity-level *Done*-to-*Executing* intrusions (if the activity has just finished executing) and special circumstance of restarts (*Executing*-to-*Executing* transitions with reset; see Section 4.1.1). At the action level, a repetition can occur through the *Done*-to-*Executing* action intrusions as long as it occurs immediately following the normative execution of that same action.

A reversal in our taxonomy occurs in the same way it would for Hollnagel's reversal (Section 4.1.2). Specifically, this occurs when an action intrusion happens where the erroneous action would normatively be the one that executes after the current normative action.

In all of these, our taxonomy does not explicitly identify that a mistimed check occurred. However, it does explicitly capture what condition received the mistimed check.

4.2.3. Discussion

As the analysis above shows, the new taxonomy is able to encompass all of the slip behaviors and thus covers the failure modes associated with slips in Reason's GEMS. As with the phenotypes of erroneous action (discussed above), many of the slip designations are associated with multiple error types in the taxonomy. Because Reason's slip designations are informal and thus have many vagaries, our new taxonomy can be seen as a useful extension of the original classification. While it is true that our taxonomy does not always explicitly capture the higher-level reason for the error (i.e. a mistimed check), it does explicitly enumerate what information was not attended to properly. Thus, our taxonomy provides additional insights into how the erroneous behavior occurred.

5. General discussion

In this work, we have introduced a new taxonomy of erroneous human behavior that classifies erroneous acts based on where and how they deviate from normative behavior expressed in task models. By linking erroneous behavior modes and/or the specific activities and actions with the information the human improperly attended to in the semantics of the task, our taxonomy links the phenotype of the erroneous behavior with its genotype (the cognitive slip). Our taxonomy is capable of accounting for the different classifications in the phenotypes of erroneous action (Hollnagel, 1993) as well as the skill-based slips from GEMS (Reason, 1990). Because these are the leading phenomenological and genotypical erroneous behavior classifications respectively, our taxonomy should be expressive enough to represent most erroneous human behaviors that occur when humans are performing task work. In some circumstances, our approach provides additional precision by explicitly enumerating why a particular erroneous phenotype occurred or exactly what information the person failed to attend to in committing a slip, thus showing that has additional expressive power beyond the original taxonomies.

Many of the categories of the legacy taxonomies such as reason's double-capture slips and Hollnagel's delays naturally arose from the erroneous sematic transitions in our taxonomy. This speaks to the validity of the new taxonomy. This is further supported by the fact that the new taxonomy encapsulated erroneous behavior types, such as post-completion errors (represented as an activity post-completion omission in our taxonomy), that have gained importance in the larger human factors literature (Bastide and Basnyat, 2007; Byrne and Bovair, 1997; Curzon and Blandford, 2004; Li et al., 2005). In fact, the taxonomy identifies a number of new and/or precise erroneous behavior types including activity completed intrusions; activity repeat-capture intrusions; activity post-repetitions; activity premature, pre-capture restarts; activity non-completions; and many different combinations of these and other types. This suggests that some or all of these special types may be deserving of additional study in future research.

The phenotypes of erroneous action are formally precise (Hollnagel, 1993). However, by design, they lack information about the factors that helped contribute the error. Conversely, the slips in GEMS (Reason, 1990) do provide information about these factors, but do so without any formal precision. Thus, our new taxonomy makes a significant contribution by connecting the phenomenological and genotypical elements of an erroneous behavior and doing so in a formal, unambiguous way. This should allow for additional precision in the study of erroneous human behavior moving forward.

Even with these contributions, the taxonomy could make further impact through future developments and applications. These are explored in the discussion below.

5.1. No-error perspective and the systemic contribution

The contemporary view of erroneous human behavior is that it is a result of system problems rather than the cause of problems (Cook et al., 1998; Dekker, 2002; Hollnagel, 1983). Our taxonomy is meant to be compatible with this perspective. Specifically, the taxonomy seeks only to classify why and how human behavior can diverge from task analytic behavior models. It does not mean to assign fault to the human operator even if he or she failed to attend to a specific part of the task or condition in the environment. In fact, the taxonomy can potentially be used as a stepping-stone towards understanding what specific systemic factors contributed to erroneous human behavior. Specifically, identifying exactly what information was not attended to, or what information was erroneously attended to, gives analysts insights into exactly what system conditions were present when the erroneous behavior occurred. Engineers could use this information to modify system behavior to avoid the associated erroneous acts. The fact that the specific violations within task model conditions provide insights into an erroneous behavior suggests that additional categorical refinements could be applied to our taxonomy. This is explored in the next section.

5.2. Additional categories

Erroneous behavior is classified hierarchically in our taxonomy (Fig. 7). This is advantageous because it allows analysts to use the taxonomy at whichever level is most useful for their purpose. As discussed previously, the erroneous behavior types can be further refined to address erroneous behaviors that arise when a person does not properly attend to specific parts of conditions. Such refinements were not included in the presented version of the taxonomy because the specific parts of violated conditions will be very specific to a given task or environmental condition under which a task occurs. The specific parts of implicit conditions (*StartConditions*, *EndConditions*, and *Resets*) that are violated will indicate exactly what part of a task's execution state was not properly attended to. The specific parts of explicit strategic knowledge conditions (*Preconditions*, *RepeatConditions*, and *CompletionConditions*) that are violated will indicate exactly what environmental conditions (based on input variables used in the condition) or memory conditions (based on local variables in the condition) were not attended to correctly. Future work should investigate if there are ways of refining the taxonomy to categorize such erroneous behaviors in ways that are useful to human factors engineers.

In the comparison of the legacy taxonomies to our new one, there are some situations where multiple erroneous behavior types are required to replicate concepts from single classifications. This suggests that there could be utility in identifying specific groupings of our erroneous behavior types to represent these behaviors. This should be investigated in future work.

5.3. Recovery behavior

Recovery behavior, where a human recognizes that he or she has made an erroneous behavior and attempts to account for it, can often be just as critical to system safety as the initiating erroneous act (Woods et al., 2010). Because recovery behaviors deviate from normative tasks,

they can be considered erroneous (Hollnagel, 1993). However, such behavior is not inherently accounted for in most erroneous behavior taxonomies (Jones, 1997) including GEMS (Reason, 1990). We know from Bigelow et al. (2011) that recovery behaviors manifest as task backtracking, restarting, resumption, or abandonment. All of these behaviors can be accounted for in our taxonomy. Backtracking occurs as any *Done*-to-*Executing* (reset) intrusion. Restarting manifests as any activity restart. Resumption occurs when a human resumes performing a task normatively after committing any erroneous behavior. Abandonment is encompassed by any activity *Executing*-to-*Done* omission. Thus, our task-based taxonomy is able to account for human recovery behavior. As such, it represents a more complete taxonomy.

5.4. Workarounds

Workarounds are human behaviors that occur when people encounter problems and find new ways to accomplish task goals. Workarounds can be both an advantage and disadvantage to a system. The possibility of working around a problem can increase system resiliency (Halbesleben et al., 2008), but also produce unexpected problems (Spear and Schmidhofer, 2005). Workarounds are not explicitly addressed in the new taxonomy. However, this does not mean that they are completely incompatible. If specific workarounds are common in a given domain, then a good task analysis should be capable of identifying them and working them into task models. In this situation, the workarounds would be normative behavior. If workaround behavior is not in the normative task model, then various intrusions should be able to capture workaround behavior. This would especially be true if the workaround behavior was a synthesis of other task behaviors. However, such a classification does not inherently capture the fact that workarounds are being developed and/or pursued. Thus, if the workaround behavior is learned later in the life of a system or developed by a human in response to novel situations, then the behavior would fall under what Reason classified as rule- or knowledge-based mistakes. These erroneous behaviors are not addressed by our taxonomy. We discuss this topic in more depth next.

5.5. Other cognitive considerations

Our taxonomy is compatible with slips from Reason's GEMS (1990). However, as discussed in Section 4.2, our taxonomy only accounts for why an erroneous behavior occurred based on what information the human operator failed to properly attend to. It does not specifically describe higher levels of classifications that relate to why the particular failures of attention could occur (such as distraction). Further, GEMS accounts for erroneous behaviors that occur at the rule-based and knowledge-based levels (mistakes). This exclusion was intentional because both types of mistakes occur when the human does not know how to perform a given task. Rule-based mistakes occur when the human has learned to do something wrong (has the wrong rule) or has problems remembering how to do the procedure at the time of execution. Knowledge-based mistakes relate to logical fallacies, incomplete knowledge, or limitations on other available information. In this context, our taxonomy would have little relevance for erroneous behaviors that manifest at the knowledge level. However, for situations where rule-based behavior can be effectively coded into task analytic models, our taxonomy could provide some relevance. In the extended literature on the use of formal verification of human-interactive systems, erroneous human behaviors have been derived from skill and rule-based behaviors represented in formal representations of cognitive architectures (Curzon and Blandford, 2004; Curzon et al., 2007; Rukšėnas et al., 2009a, 2007, 2009b). This suggests rule-based mistakes and high classifications of skill-based slips could be reconciled with our taxonomy. Future work should investigate how our method could be adapted to account for these classifications.

5.6. Generalizability

EOFM, the task analytic modeling system around which the new taxonomy is based, is unique within the formal task model community. Specifically, EOFM is automata-based while other common task modeling formalisms such as CTT (Paternò et al., 1997) and the system used by Fields (2001) are based on process algebras. While they have similar expressive power, automata and process algebras are definitely different. Thus, it is not clear whether the classifications contained in our taxonomy will be easily translated to other process-algebra-based task models. However, EOFM and these other languages have similar expressive power. Thus, it does seem like it is possible to reconcile the task-based taxonomy with these other task models. This should be the subject of future work.

5.7. Cognitive work analysis

Hierarchical task models, like those used as the basis for our taxonomy, are widely used in the human factors engineering community. However, these models are not without their critics. In particular, models like EOFM are "instruction-based" approaches. These work well in situations where people have well defined plans or what Vicente (1999) would call "closed" systems. However, they do not give workers very much flexibility or discretion when interacting with a complex "open" system (Vicente, 1999). As such, cognitive work analysis (CWA) (Vicente, 1999) has been defined as a method for characterizing the constraints on human work in a way that is appropriate for both "closed" and "open" systems. Extensive work has gone into developing this approach and adapting it for use in a number of domains (Bisantz and Burns, 2008), including work within the formal methods community (Masci et al., 2011; Wright et al., 2000). Ultimately, there are tradeoffs between CWA and task analysis (Salmon et al., 2010), and they are not necessarily incompatible. While CWA can capture more information (at higher levels abstraction), task analysis is more detailed. Ultimately, because task analysis is more widely used, our taxonomy should have utility in the human factors community. This is further bolstered by the fact that task analysis can be used in a number of other analyzes (Salmon et al., 2010). This is explored next.

5.8. Application areas and use in analyzes

A taxonomy inherently has intellectual value if it provides a unique perspective on a subject, something we believe our taxonomy accomplishes. However, our task-based erroneous human behavior taxonomy also has the potential to have impact through its use in a number of application areas. We explore several of these below.

5.8.1. Erroneous behavior tracking at runtime

The phenotypes of erroneous behavior were originally intended for use in monitoring to identify erroneous behavior at runtime (Hollnagel and Marsden, 1996). Such functionality can be used as the basis for intelligent decision support systems (Guerlain et al., 1999; Hollnagel and Marsden, 1996). There is also precedence for creating monitors for tracking human task behavior at runtime (Bushman et al., 1993; Chu et al., 1995; Thurman et al., 1998). As such, the taxonomy enumerated here should be readily adaptable to the detection of erroneous behavior at runtime. Because our taxonomy contextualizes the actual erroneous behavior manifestation with environmental information contained in strategic knowledge conditions, it should provide more information than simply detecting and identifying the behavior's erroneous phenotype.

5.8.2. Accident analysis and reporting

GEMS has served as the basis for erroneous human behavior classification in accident analyzes and event reporting. This includes the Human Factors Analysis and Classification System (which is generic but

predominately used in aviation; Shappell and Wiegmann, 2000) and Zhang et al.'s cognitive taxonomy of medical errors (Zhang et al., 2004). Given that our taxonomy adds additional precision to the skill-level erroneous behaviors of these systems and the precedence task models have in accident analysis (Doytchev and Szwillus, 2009; Hollnagel, 1998), our new taxonomy could be used in accident analysis and reporting. Further, the fact that our taxonomy uses formal models of human operator tasks and provides formal descriptions of erroneous human behaviors should potentially make it compatible with techniques for formally modeling and reasoning about accidents (Johnson, 1997; Johnson and Holloway, 2003; Johnson and Telford, 1996). This should be investigated in future work.

5.8.3. Human reliability analysis

Because the model presented here is a taxonomy, it only strives to classify erroneous human behavior, not address the plausibility of different behaviors. This is purview of human reliability analyzes ([Bell and Holroyd, 2009]). Human reliability analyzes attempt to qualitatively and/or quantitatively assess the probability of erroneous human behavior. There are a number of such techniques including THERP (technique for human error-rate prediction; Swain and Guttmann, 1983); CREAM (cognitive reliability and error analysis method; Hollnagel, 1998); the data entry error rate prediction method developed by Cauchi (2013); and the HAZOP-like techniques such as THEA (Pocock et al., 2001a; 2001b) and the approach explored by Paternò and Santoro (2002). While all of these techniques have different underlying theories and procedures for estimating error rates (i.e. historical data, cognitive control modes, brainstorming activities), they are similar in that they consider deviations from human operator tasks. Because our taxonomy is based on systematic deviations from normative task models, it should provide a good framework around which to assess human reliability. Thus, future work should investigate how our taxonomy can be integrated into or help extend existing human reliability analyzes.

5.8.4. System design

User-centered design is a framework for designing human-computer or human-machine interfaces so that they always support the human operators' tasks (Vredenburg et al., 2002). Erroneous human behavior is an extremely important consideration in user-centered design. This is because designers want to design interfaces that prevent or avoid certain erroneous acts and reduce the likelihood of others (Rizzo et al., 1996). The task-based nature of our erroneous behavior taxonomy should make it compatible with user-centered design. The connection the taxonomy provides between the phenotype and genotype of erroneous acts should facilitate designs that both prevent them and help humans avoid the associated cognitive failures. Future research should explore how our taxonomy can be integrated into user-centered design practices.

5.8.5. Simulation and formal verification

Task analytic models have been used in model-based analyzes such as simulation and formal verification to assess the safety and reliability of human-interactive systems both with and without erroneous human behavior (see Bolton et al., 2013). In particular, Bastide and Basnyat (2007) and Fields (2001) showed how patterns of erroneous human behavior, mostly based on the phenotypes of erroneous behavior, could be manually used to modify normative human task behavior. The impact this behavior has on system performance can then be assessed using simulation and/or formal verification analyzes. Conversely, Bolton et al. have explored how erroneous human behavior can be automatically generated from task models by systematically considering the performance of erroneous phenotypes during the performance of actions (Bolton et al., 2012), slips caused by attentional failures to strategic knowledge (Bolton and Bass, 2013), miscommunications in communication protocols (Bolton, 2015), and the combination of all of the above (Pan and Bolton, 2016). Although based on similar foundations, the erroneous behavior taxonomy discussed here is far more complete than any of these other analyzes, even in combination. As such, the new taxonomy should be capable of being used to generate erroneous human behavior in simulation and formal verification analyzes. By virtue of its completeness, an erroneous behavior generation technique based on the taxonomy should help analysts evaluate system safety at a level that was previously not possible. This should be investigated in future work.

6. Conclusions

By unifying the phenomenological and genotypical perspectives on erroneous human behavior, our task-based taxonomy constitutes a significant contribution. As we have shown, this new taxonomy is compatible with the two leading erroneous behavior taxonomies and thus supports the different classifications they encompass. In addition to organically accounting for common erroneous behaviors (such as capture and post-completion errors), our taxonomy identifies a number of new erroneous behavior types that should be given consideration by the larger human factors community. Finally, the fact that the taxonomy is compatible with the legacy system and contextualizes erroneous behaviors around task analytic models makes it compatible with many facets of human factors engineering and analysis. As such, the true impact of the taxonomy on system safety and usability will be realized as it is adopted and incorporated into these different practices.

Acknowledgments

The work presented here was supported by grant W911NF-15-1-0474 "Young Investigator Program (8.5): Preventing Complex Failures of Human Interactive Systems with Erroneous Behavior Generation and Robust Human Task Behavior Patterns" by the Army Research Office / Army Research Lab. The author would like to thank Kylie Molinaro and Adam Houser for their help in the preparation of this manuscript.

References

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.

Bell, J., Holroyd, J., 2009. Review of Human Reliability Assessment Methods. Technical Report, RR679. Health and Safety Executive, Norwich.

Bigelow, M., Christmann, C., Feigh, K., Prichett, A., Kannan, S., Kim, S.-Y., Lee, G., 2011. WMC. Technical Report. GA Tech, Atlanta.

Bisantz, A.M., Burns, C.M., 2008. Applications of Cognitive Work Analysis. CRC Press, Boca Raton.

Bolton, M.L., 2015. Model checking human–human communication protocols using task models and miscommunication generation. J. Aerosp. Inf. Syst. 12 (7), 476–489.

Bolton, M.L., Bass, E.J., 2008. Formal modeling of erroneous human behavior and its implications for model checking. In: Proceedings of the Sixth NASA Langley Formal Methods Workshop. NASA Langley Research Center, Hampton, pp. 62–64.

Bolton, M.L., Bass, E.J., 2013. Generating erroneous human behavior from strategic knowledge in task models and evaluating its impact on system safety with model checking. IEEE Trans. Syst. Man Cybern. 43 (6), 1314–1327.

Bolton, M.L., Bass, E.J., 2017. Enhanced Operator Function Model (EOFM): A Task Analytic Modeling Formalism for Including Human Behavior in the Verification of Complex Systems. In: Weyers, B., Bowen, J., Dix, A., Palanque, P. (Eds.), The Handbook of Formal Methods in Human-Computer Interaction. Springer International Publishing, Cham, pp. 343–377. doi:10.1007/978-3-319-51838-1_13.

Bolton, M.L., Bass, E.J., Siminiceanu, R.I., 2012. Generating phenotypical erroneous human behavior to evaluate human-automation interaction using model checking. Int. J. Hum. Comput. Stud. 70 (11), 888–906.

Bolton, M.L., Bass, E.J., Siminiceanu, R.I., 2013. Using formal verification to evaluate human-automation interaction in safety critical systems, a review.. IEEE Trans. Syst. Man Cybern. 43 (3), 488–503.

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

Bolton, M.L., Zheng, X., Molinaro, K., Houser, A., Li, M., 2016. Improving the scalability of formal human–automation interaction verification analyzes that use task-analytic models. Innov. Syst. Softw. Eng. DOI: 10.1007/s11334-016-0272-z.

Bushman, J.B., Mitchell, C.M., Jones, P.M., Rubin, K.S., 1993. ALLY: an operator's associate for cooperative supervisory control systems. IEEE Trans. Syst. Man Cybern. 23 (1), 111–128.

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

Cauchi, A., 2013. Using differential formal analysis for dependable number entry. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, pp. 155–158.
Chu, R.W., Mitchell, C.M., Jones, P.M., 1995. Using the operator function model and OFMspert as the basis for an intelligent tutoring system: towards a tutor/aid paradigm for operators of supervisory control systems. IEEE Trans. Syst. Man Cybern. Part A 25 (7), 1054–1075.
Cook, R.I., Woods, D.D., Miller, C., 1998. A Tale of Two Stories: Contrasting Views of Patient Safety. Technical Report. National Health Care Safety Council of the National Patient Safety Foundation at the AMA.
Cooley, M., 2000. Human-centered design. Inf. Des. 59–81.
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 Comput. 19 (4), 513–550.
Dekker, S.W.A., 2002. The Re-invention of Human Error. Technical Report, 2002-01. Lund University School of Aviation, Ljungbyhed, Sweden.
Doytchev, D.E., Szwillus, G., 2009. Combining task analysis and fault tree analysis for accident and incident analysis: a case study from Bulgaria. Accid. Anal. Prev. 41 (6), 1172–1179.
Dunjó, J., Fthenakis, V., Vílchez, J.A., Arnaldos, J., 2010. Hazard and operability (HAZOP) analysis. a literature review. J. hazard. Mater. 173 (1), 19–32.
Fields, B., Harrison, M., Wright, P., 1997. THEA: Human Error Analysis for Requirements Definition. Technical Report. York.
Fields, R.E., 2001. Analysis of Erroneous Actions in the Design of Critical Systems. University of York, York Ph.D. thesis.
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 Egnineering. Springer, Berlin, pp. 98–109.
Guerlain, S.A., Smith, P.J., Obradovich, J.H., Rudmann, S., Strohm, P., Smith, J.W., Svirbely, J., Sachs, L., 1999. Interactive critiquing as a form of decision support: an empirical evaluation. Hum. Factors 41 (1), 72–89.
Halbesleben, J.R., Wakefield, D.S., Wakefield, B.J., 2008. Work-arounds in health care settings: literature review and research agenda. Health Care Manage. Rev. 33 (1), 2–12.
Hollnagel, E., 1983. Position paper on human error. In: NATO Conference on Human Error. Bellagio, Italy.
Hollnagel, E., 1993. The phenotype of erroneous actions. Int. J. Man Mach. Stud. 39 (1), 1–32.
Hollnagel, E., 1998. Cognitive Reliability and Error Analysis Method (CREAM). Elsevier, Amsterdam.
Hollnagel, E., Marsden, P., 1996. Further Development of the Phenotype Genotype Classification Scheme for the Analysis of Human Erroneous Actions. Technical Report, EUR 16463 EN. European Commission.
Johnson, C.W., 1997. The epistemics of accidents. Int. J. Hum. Comput. Stud. 47 (5), 659–688.
Johnson, C.W., Holloway, C.M., 2003. Strengths and weaknesses of logic formalisms to support the causal analysis of mishaps. In: Proceedings of the 21st International System Safety Conference. International Systems Safety Society, Unionville, pp. 1133–1142.
Johnson, C.W., Telford, A.J., 1996. Extending the application of formal methods to analyse human error and system failure during accident investigations. Softw. Eng. J. 11 (6), 355–365.
Jones, P.M., 1997. Human error and its amelioration. In: Handbook of Systems Engineering and Management. Wiley, pp. 687–702.
Kebabjian, R., 2016. Accident Statistics. planecrashinfo.com, Accessed 3/14/2016
Kenny, D.J., 2015. 24th Joseph T. Nall report: General Aviation Accidents in 2012. Technical Report. AOPA Foundation.
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.
Lawley, H.G., 1974. Operability studies and hazard analysis. Chem. Eng. Prog. 70 (4), 45–56.
Li, M., Molinaro, K., Bolton, M.L., 2015. Learning formal human-machine interface designs from task analytic models. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting. HFES, Santa Monica, pp. 652–656.
Li, M., Wei, J., Zheng, X., Bolton, M.L., 2017. A formal machine learning approach to generating human-machine interfaces from task models. IEEE Trans. Hum. Mach. Syst. doi:10.1109/THMS.2017.2700630. In Press.
Li, Y.W., Blandford, A., Cairns, P., Young, R.M., et al., 2005. Post-completion errors in problem solving. In: Proceedings of the XXVII Annual Conference of the Congnitive Science Society. Cognitive Science Society, Inc., Wheat Ridge, pp. 1278–1283.
Manning, S.D., Rash, C.E., LeDuc, P.A., Noback, R.K., McKeon, J., 2004. The Role of Human Causal Factors in US Army Unmanned Aerial Vehicle Accidents. Technical Report, 2004-11. USA Army Research Laboratory.
Martinie De Almeida, C., Palanque, P., Ragosta, M., Fahssi, R.M., 2013. Extending procedural task models by explicit and systematic integration of objects, knowledge and information. In: 31st European Conference on Cognitive Ergonomics. ACM.
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 Trans. Syst. Man Cybern. Part A 16 (3), 343–357.
NHTSA, 2008. National Motor Vehicle Crash Causation Survey: Report to Congress. DOT HS 811 059.
Norman, D.A., 1988. The Psychology of Everyday Things. Basic Books, New York.
Office of Technology Assessment, 1993. Who Goes There: Friend or Foe. Technical Report, OTA-ISC-537. Congress, US, Washington, DC.
Pan, D., Bolton, M.L., 2016. Properties for formally assessing the performance level of human-human collaborative procedures with miscommunications and erroneous human behavior. Int. J. Ind. Ergon. doi:10.1016/j.ergon.2016.04.001. In Press.
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, London, pp. 362–369.
Paternò, F., Santoro, C., 2002. Preventing user errors by systematic analysis of deviations from the system task model. Int. J. Hum. Comput. Stud. 56 (2), 225–245.
Perrow, C., 1999. Normal Accidents: Living with High-risk Technologies. Princeton University Press, Princeton.
Pocock, S., Fields, B., Harrison, M., Wright, P., 2001a. THEA: A Reference Guide. Technical Report, YCS336. Department of Computer Science, University of York.
Pocock, S., Harrison, M.D., Wright, P.C., Johnson, P., 2001b. THEA: a technique for human error assessment early in design.. In: Interact. IFIP Technical Committee No 13 on Human-Computer Interaction, Tokyo, pp. 247–254.
Rasmussen, J., 1983. Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. IEEE Trans. Syst. Man. Cybern. 13 (3), 257–266.
Reason, J., 1990. Human Error. Cambridge University Press, New York.
Rizzo, A., Parlangeli, O., Marchigiani, E., Bagnara, S., 1996. The management of human errors in user-centered design. SIGCHI Bull. 28 (3), 114–118.
Rukšėnas, R., Back, J., Curzon, P., Blandford, A., 2009a. Verification-guided modelling of salience and cognitive load. Formal Aspects Comput. 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.
Rukšėnas, R., Curzon, P., Blandford, A., Back, J., 2009b. Combining human error verification and timing analysis. In: Proceedings of the 2007 Conferences on Engineering Interactive Systems. Springer, Berlin, pp. 18–35.
Salmon, P., Jenkins, D., Stanton, N., Walker, G., 2010. Hierarchical task analysis vs. cognitive work analysis: comparison of theory, methodology and contribution to system design. Theor. Issues Ergon. Sci. 11 (6), 504–531.
Schraagen, J.M., Chipman, S.F., Shalin, V.L., 2000. Cognitive Task Analysis. Lawrence Erlbaum Associates, Inc., Philadelphia.
Shappell, S., Wiegmann, D., 2000. Human Factors Analysis and Classification System-HFACS. Technical Report, DOT/FAA/AM-00/7. Office of Aviation Medicine, Washington, DC.
Sheridan, T.B., Parasuraman, R., 2005. Human-automation interaction. Rev. Hum. Factors Ergon. 1 (1), 89–129.
Spear, S.J., Schmidhofer, M., 2005. Ambiguity and workarounds as contributors to medical error. Ann. Intern. Med. 142 (8), 627–630.
Swain, A.D., Guttmann, H.E., 1983. Handbook of Human-Reliability Analysis with Emphasis on Nuclear Power Plant Applications. Final report. Technical Report NUREG/CR-1278; SAND-80-0200 ON: DE84001077. Sandia National Labs, Albuquerque.
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.
Vicente, K.J., 1999. Cognitive Work Analysis: Toward Safe, Oroductive, and Healthy Computer-based Work. Lawrence Erlbaum Associates, Inc., Philadelphia.
Vredenburg, K., Mao, J.-Y., Smith, P.W., Carey, T., 2002. A survey of user-centered design practice. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 471–478.
Woods, D.D., Dekker, S., Cook, R., Johannesen, L., Sarter, N., 2010. Behind Human Error. Ashgate Farnham.
Wright, P.C., Fields, R.E., Harrison, M.D., 2000. Analyzing human-computer interaction as distributed cognition: the resources model. Hum. Comput. Interact. 15 (1), 1–41.
Zhang, J., Patel, V.L., Johnson, T.R., Shortliffe, E.H., 2004. A cognitive taxonomy of medical errors. J. Biomed. Inform. 37 (3), 193–204.