A Formal Methods Approach for Predicting How Users will Utilize System Features

Meng Li, Sara Behdad, and Matthew L. Bolton
University at Buffalo, the State University of New York

Model checking is increasingly being used with task analytic behavior models to prove whether models of human-interactive systems are safe and reliable. Such methods could be used to predict how different types of users will choose to use system features. However, existing methods focus on modeling the full space of possible human behaviors without considering how users will choose to navigate this space. In this work, we present a new approach that enables model checking to predict how different types of users will use features of an interactive system by employing a novel combination of task analytic modeling and utility theory. This paper presents this method and illustrates its power with a smart thermostat application. The results of the application analysis and its implications for future research are discussed.

INTRODUCTION

Formal methods (Wing, 1990) are tools and techniques from computer science that have been increasingly finding applications in human factors engineering (Bolton, 2017a; Bolton, Bass, & Siminiceanu, 2013; Weyers, Bowen, Dix, & Palanque, 2017). This is because formal methods provide robust, mathematical tools, such as model checking (Clarke, Grumberg, & Peled, 1999), that prove whether system models satisfy specifications. For human factors, this can be used to determine if a system design satisfies system performance or usability requirements. In particular, a number of researchers have investigated how task analytic behavior models can be included in formal systems analyses to find human-automation interaction issues that could cause problems (Bolton, 2017a; Bolton et al., 2013). These techniques are very powerful for safety analyses because they account for all of the ways that human task behavior can be performed. However, they do not consider how and why different types of users will use systems differently. If individual preferences were included, model checking could be used to explore how types of users will use system features.

We address this here by combining formal task analytic techniques with utility theory concepts. This allows us to use model checking to predict how different users will employ system features. Below we cover the background necessary for understanding our approach. We then describe our method and illustrate its power by using it to analyze a smart thermostat.

BACKGROUND

Formal Methods and Model Checking

Formal methods are tools and techniques for specifying, modeling, and verifying systems (Wing, 1990). Formal models describe the behavior of the system being evaluated. Specification properties mathematically describe desirable system qualities. Formal verification is the process of mathematically proving whether the systems always satisfies a specification.

Model checking is a fully automated process to formal verification (Clarke et al., 1999). In this, a formal model represents the system as a collection of concurrently executing finite state machines: variables and transitions between variable values (states). Specifications use temporal logic (Emerson, 1990) to assert desirable system properties using model variables, Boolean operators, and temporal operators. Formal verification is performed by exhaustively searching through the entire statespace of the system model (often using extremely statespace-efficient algorithms) to determine if a specification holds. If it does, the model checker has proven that the system model satisfies the specification. If it does not, a counterexample is returned. This shows exactly how the specification was violated as a trace through the system model's statespace.

Model checking is widely used in the verification of computer software and hardware (Grumberg & Veith, 2008). It has also been used increasingly in the human-machine interaction community (Bolton, 2017a; Bolton et al., 2013). Of particular relevance to the presented work is the research focused on formal verification with task analytic models.

Model Checking with Task Analytic Models

Task analytic behavior models (or task models) are produced by a task analysis (Kirwan & Ainsworth, 1992) and represent the behaviors humans use for achieving goals with a system. These are widely used in human factors to design interfaces (Li, Wei, Zheng, & Bolton, 2017; Paternò, 2000), develop training (Chu, Mitchell, & Jones, 1995), and drive usability analyses (Lecerof & Paternò, 1998).

Model checking is good at finding unexpected systems interactions. Thus, researchers have explored its use in finding human-automation issues that manifest between models of system behavior and human task behavior (Bolton, 2017a; Bolton et al., 2013). This includes evaluating the effect normative behavior has on system safety (Aït-Ameur & Baron, 2006; Bolton, Siminiceanu, & Bass, 2011; Paternò & Santoro, 2001) as well as included (Bastide & Basnyat, 2007; Fields, 2001) or generated (Bolton, 2015, 2017b; Bolton & Bass, 2013; Bolton, Bass, & Siminiceanu, 2012; Pan & Bolton, 2018) erroneous behavior.

In these analyses, the power comes from the model checker exploring all of the allowable human behaviors in all of the contexts where they can possibly occur. This makes them particularly good at finding unexpected interactions and system failures. However, a side effect of this is that the analyses do not account for the particular ways that different types of humans may use a given device. This means that the formal methods are ill suited to exploring how demographics of people will choose to interact with a device or use different features. Below we discuss the modeling of human decision-making with utility theory, a technology capable of addressing this issue.

Utility Theory

Utility theory is employed readily in economics and the behavioral sciences. It models human preferences and decisions using a utility or value function (Von Neumann & Morgenstern, 1953). This function dictates preference order between available options based on how the decision maker thinks options will achieve his or her goals. As a quantitative tool, a utility or value function encapsulates attributes of alternatives and synthesizes them into single values that can be compared: the higher the value, the more preferable. The overall value/utility of an alternative depends on the combination of the attribute levels. These levels are standardized and assigned scaling factors that correspond to their relative importance to the decision-maker, reflecting their tradeoff in the decision-making process. The value function can be expressed in unit-less utility or other units that have explicit value (such as dollars).

Utility theory has been used to model human decision-making in a number of different economic and non-economic applications (Gintis, 2014). It has also been used in formal verification analyses. However, these applications are predominantly used to prove that self-adapting automation, which uses utility functions as part of its process, will not reach undesirable states (Calinescu & Kwiatkowska, 2009; Cámara, Garlan, Schmerl, & Pandey, 2015; Lomuscio, Strulo, Walker, & Wu, 2010; Sykes, Heaven, Magee, & Kramer, 2010). To the best of our knowledge, nobody has used utility theory to model human decision-making in a formal verification context.

OBJECTIVE

In this work, we set out to enable formal verification analyses to reason about how different classes of users will use device features. To accomplish this we introduce the ability to model value functions from utility theory within EOFM. This enables analysts to describe how different human operators will assign values to different decision-making options. Modeled users will choose between options based on which produces the highest value. In the following, we describe how this capability was incorporated into EOFM and its supported formal verification analyses. We illustrate how this can be used to reason about the feature utilization by evaluating a smart thermostat application.

METHOD

The Enhanced Operator Function Model

In this work, we use the Enhanced Operator Function Model (EOFM), a formal task modeling language (Bolton & Bass, 2009, 2017). EOFM is XML-based and represents human task behavior as an input/output system. Inputs (which are variables) come from parts of the system that are interacting with the human, such as the human-machine interface and the environment. Outputs are human actions. The task model describes how human actions are generated based on input variables (representing human-observable phenomena from other parts of the system) and local variables (representing perceptual or cognitive phenomena). Types and constants can also be defined within an EOFM to declare variables and drive task behavior. The relationships between these elements are shown in Figure 1.

An EOFM task is a hierarchy of activities and actions. Activities decompose into other activities or, at the lowest level of the hierarchy, atomic actions. A decomposition operator describes how activities or actions in a decomposition execute. EOFM has nine such operators (see Bolton et al. 2011). In this work, we use two: *ord* and *xor*. With an *ord* operator, all activities or actions must execute in the order they appear in the decomposition. In an xor decomposition, exactly one activity or action executes. Strategic knowledge associated with activities is represented with Boolean expressions using constants, input variables, and local variables. These assert activity preconditions, repeat conditions, and completion conditions.

EOFMs can be rendered visually as tree-like graphs (shown later in Figure 3). Actions are rectangles and activities are rounded rectangles. A decomposition is an arrow with the operator as its label. The arrow points to a rounded rectangle containing the decomposed acts. Strategic knowledge conditions connect to their associated activity with the condition logic as the label. A *Precondition* is a downward-pointing, yellow triangle; a *CompletionCondition* is an upward-pointing magenta triangle; and a *RepeatCondition* is a recursive arrow.

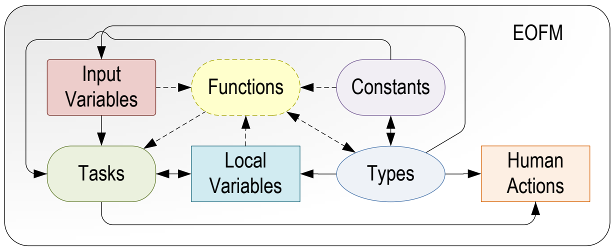

Figure 1. The different elements of an EOFM (including the new functions) and how they interrelate. Types can be specified by analysts (or be inherent to the EOFM language: Boolean, integer, real) optionally using functions and constants. Input variables, local variables, constants, functions, and human actions have types. Task behavior can use input variables, local variables, constants, and functions during task execution. It is capable of modifying human actions (outputs) and local variables (for cognitive or perceptual actions).

EOFMs have formal semantics that describe how they execute as finite state machines (see Bolton et al. 2011; Bolton, Zheng, Molinaro, Houser, and Li 2017). These formal semantics allow EOFMs to be used in model checking analyses. Specifically, EOFM has a Java-based translator that converts an EOFM into the input language of SAL (de Moura, Owre, & Shankar, 2003). This allows EOFM to be used as part of a larger system model to verify system safety properties.

EOFM has been used in a number of model-checking-based formal verification analyses (see Bolton and Bass 2017). However, it previously lacked the mechanisms for dynamically translating inputs, local variables, and constants with a value function and have this impact task performance.

Extending EOFM with Value Functions

In this work, we extended EOFM to enable the inclusion of value functions so that they can be used to affect the execution of task behavior. To accomplish this, we incorporated a function element into EOFM (see Figure 1). Mathematically, functions are constant mappings between the types. In fact, formal modeling languages like those supported by SAL (de Moura et al., 2003) represent functions in this way. Thus, functions were implemented in EOFM by extending the constant element. Now within the EOFM xml code, an analyst can create functions by giving a constant parameters and specifying the function's mathematical transformation in the constant's value.

By giving EOFM the ability to represent functions, functions can influence how tasks execute. This can occur in two ways. In the first, functions can be used to assign values to variables at the human action level, either to outputs via human actions or to represent cognitive or perceptual actions through assignments to local variables. In the second, functions can be called as part of a strategic knowledge condition to help determine when activities can execute.

This second approach specifically relates to how we use utility theory in our new approach. Specifically, an analyst can formulate a value function using the new function construct. Then, this value function can be called in the preconditions of task model activities to determine if specific activities will execute based on the conditions and options available to the human at the time of task execution.

Figure 2. The smart thermostat application evaluated with our new approach.

With these changes, EOFM's analysis approach (where the task model is translated into a larger formal system model describing the behavior of other system elements; Bolton and Bass 2009, 2017) can be used to reason about how humans with particular value functions will use system features. Below we show how this can be done with a smart thermostat application.

APPLICATION

In our smart thermostat application (Figure 2), the user interacts with the device by either scrolling (rotating the silver ring) and clicking (by pushing and releasing the ring). Users can use the device to set the temperature. However, most importantly to this work, users can enable and disable the device's smart "auto-away" feature. When enabled, this uses the device's sensors to determine when users are away and adjusts the heat to save money. When a user returns home, the device brings the temperature back up to the user's selected nominal value.

For the purposes of this analysis, we assume that analysts have used demographic research to identify the value functions of two groups of consumers who might use this device. They are interested in determining (based on the range of temperatures experienced during the winter in the given area) whether or not the two groups would use the auto-away feature.

Below we describe the value function and how it was incorporated into the application's model.

Value Function, Task, and System Modeling

The value function in this application is used to determine if the user wants to enable or disable the auto-away feature based on the relative tradeoffs between the level of comfort and the cost of heating associated with feature use during the cold season. To address this, the task behavior model takes an input (*iEnvironmentTemp*) representing the temperature of the environment outside the house as a real number. There are also two constants ($cAutoTemp = 50$ and $cIdealTemp = 71$) representing the minimum temperature (in degrees Fahrenheit) auto-away will maintain and the ideal temperature for the human user (respectively). Several functions are used to ultimately compute quantities that will be of use to the value function.

The first determines what the indoor temperature will be based on temperature that will be set by the user as the intended indoor temperature (S) and the temperature outside (E) as:

$$cIndoorTemp(S,E) = \begin{cases} E & \text{if } E > S \\ S & \text{otherwise.} \end{cases} \quad (1)$$

The second (based on the average cost of heating the houses of the target demographic up one degree) computes the cost to heat the house to the indoor temperature from the outside temperature (E):

$$cCost(S,E) = \$0.015 \cdot (cIndoorTemp(S,E) - E). \quad (2)$$

The third (based on the levels of comfort from the international standard; ISO 2005) represents the value users place on coming home to a house at the indoor temperature (values are shown for both versions of the utility functions in the parentheses):

$$cComfort(S,E) = \begin{cases} \$(0.10, 1.00) & \text{if } 67 \le cIndoorTemp(S,E) < 75 \\ \$(0.05, 0.50) & \text{if } 64 \le cIndoorTemp(S,E) < 67 \\ \$(0.01, 0.10) & \text{if } 61 \le cIndoorTemp(S,E) < 64 \\ \$(0.00, 0.00) & \text{otherwise.} \end{cases} \quad (3)$$

Finally, the value function for deciding whether to enable or disable auto-away is computed based on the value of comfort (V; calculated using Equation (3)) and the cost (C; calculated using Equation (2)) associated with a given choice:

$$cVal(S,E) = cComfort(S,E) - cCost(S,E). \quad (4)$$

These constants and functions are used by the EOFM task to determine whether or not to turn the auto-away feature off or on (Figure 3). This is represented in the precondition of (*aChangeAwaySetting*). The user will perform the task if auto-away is off and the value of turning it on is higher than the value of leaving it off. The user will also perform the task if auto-away is on and the value of turning it off is higher than the value of leaving it on. To make the change, the user first exits the temperature display (*aExitTempDisplay*) by performing the click action (*hClick*). This takes them to the main menu where icons of various features are listed. The user then navigates through the icons (*aScrollToAutoAway*) using the scroll action (*hScroll*) until the auto-away icon is selected. By clicking, the user opens the auto-away menu (*aEnterAwayMenu*). Then, the user (under *aSwitchAutoAwaySetting*) scrolls the auto-away options to switch the feature from off to on or from on to off. Finally, the user confirms the change and returns to the main menu (*aConfirmChange*) by clicking.

We also modeled the human operator's task for returning to the temperature display (though this is not reported).

The EOFM task model was translated into the input language of SAL using the state-space-optimized EOFM translator (Bolton et al., 2017). This was paired with a formal model that represented how the thermostat responded to user inputs. It also represented the outside temperature (*iEnvironmentTemp*) as an open parameter that could be any real valued Fahrenheit temperature between 0 and 60 degrees.

Model Checking and Results

In these analyses, we wanted to determine if the user would ever enable or disable auto-away. This was accomplished by checking two specification properties. The first asserts that auto-away will never be turned off. This used linear temporal logic to assert that globally it should never be true that auto-away will be on and eventually off:

$$\mathbf{G}\neg(iAutoAway = On \wedge \mathbf{F}(iAutoAway = Off)). \quad (5)$$

The second asserts that auto-away will never be turned on:

$$\mathbf{G}\neg(iAutoAway = Off \wedge \mathbf{F}(iAutoAway = On)). \quad (6)$$

Each property was checked against two versions of the formal model. Each model was identical except for the variation in the value function resulting from the value options in Equation (3). Model checking was completed using SAL's infinite bounded model checker (de Moura et al., 2004) using a depth of 30 and the iterative search option. This was performed on a computer workstation with a 3.60 GHz Intel® Xeon® E5-1650 CPU with 128 Gigabytes of RAM running Linux Mint.

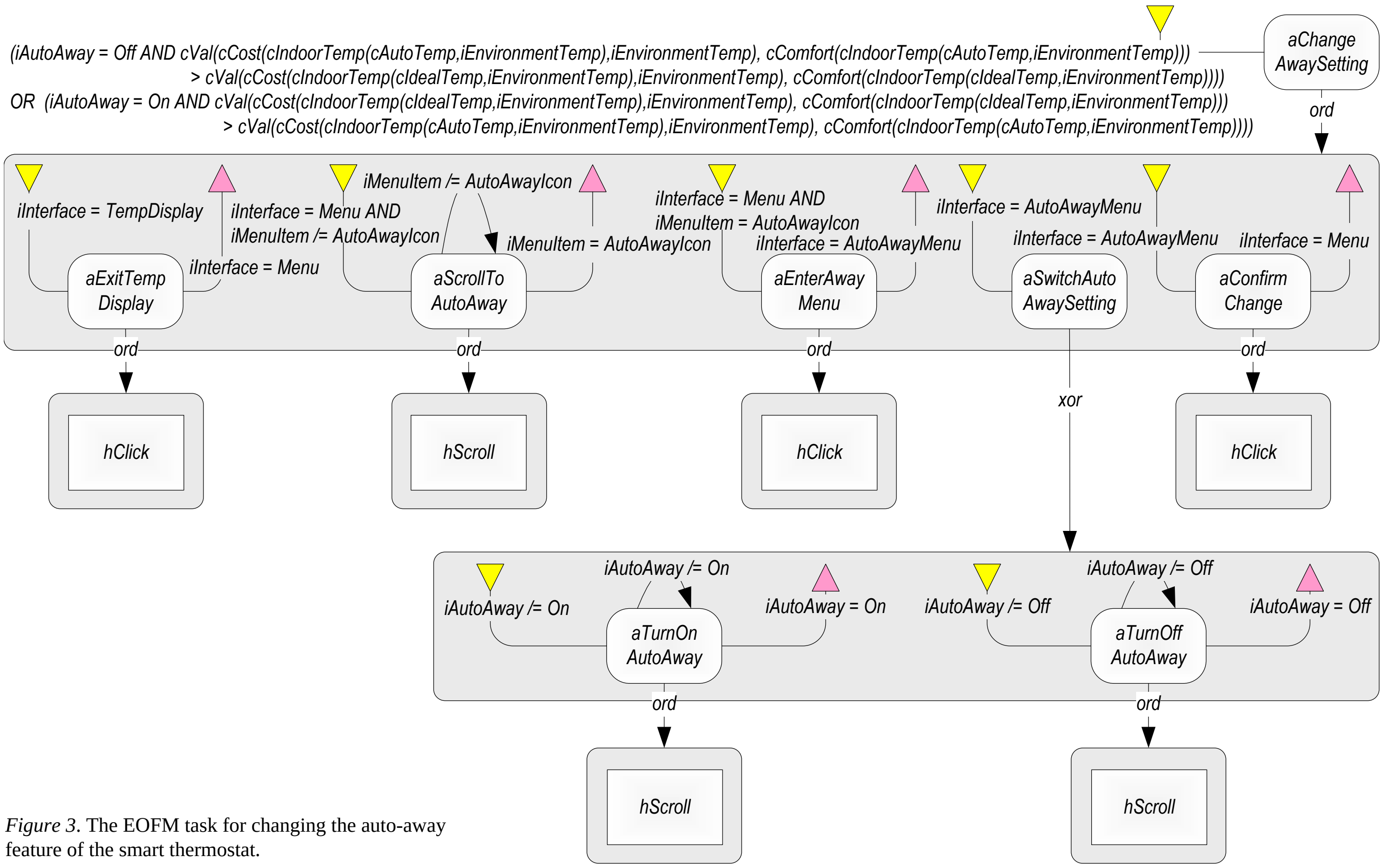
(iAutoAway = Off AND cVal(cCost(cIndoorTemp(cAutoTemp,iEnvironmentTemp),iEnvironmentTemp), cComfort(cIndoorTemp(cAutoTemp,iEnvironmentTemp)))
> cVal(cCost(cIndoorTemp(cIdealTemp,iEnvironmentTemp),iEnvironmentTemp), cComfort(cIndoorTemp(cIdealTemp,iEnvironmentTemp))))
OR (iAutoAway = On AND cVal(cCost(cIndoorTemp(cIdealTemp,iEnvironmentTemp),iEnvironmentTemp), cComfort(cIndoorTemp(cIdealTemp,iEnvironmentTemp)))
> cVal(cCost(cIndoorTemp(cAutoTemp,iEnvironmentTemp),iEnvironmentTemp), cComfort(cIndoorTemp(cAutoTemp,iEnvironmentTemp))))
aChange AwaySetting
ord
iInterface = TempDisplay
iInterface = Menu AND iMenuItem /= AutoAwayIcon
iInterface = Menu
aExitTemp Display
iMenuItem /= AutoAwayIcon
iMenuItem = AutoAwayIcon
aScrollTo AutoAway
iInterface = Menu AND iMenuItem = AutoAwayIcon
iInterface = AutoAwayMenu
aEnterAway Menu
iInterface = AutoAwayMenu
aSwitchAuto AwaySetting
iInterface = AutoAwayMenu
iInterface = Menu
aConfirm Change
ord
hClick
hScroll
hClick
xor
hClick
iAutoAway /= On
iAutoAway /= On
aTurnOn AutoAway
iAutoAway = On
iAutoAway /= Off
iAutoAway /= Off
aTurnOff AutoAway
iAutoAway = Off
hScroll
hScroll

Figure 3. The EOFM task for changing the auto-away feature of the smart thermostat.

All verifications took less than 22 seconds.

For the first model (with the more modest values associated with user comfort), the specification in Equation (5) proved to be true and the specification in Equation (6) produced a counterexample. This means that this type of user will always try to turn the feature on and, if auto-away is on, it will never be turned off. For the second model (where users place a higher value on their comfort), the specification in Equation (5) produced a counterexample and the specification in Equation (6) proved to be true. This means that this type of user will always try to turn the feature off and, if auto-away is off, it will never be turned on.

DISCUSSION

This work has introduced a new method for integrating value function concepts from utility theory into EOFM tasks. This allows task analytic modeling concepts to be used synergistically with utility-based decision theory. In particular, in this work we showed how this could be used to predict how different types of users will exploit features in a device.

The results of the smart thermostat case study demonstrate the capabilities of this approach. We found that one user demographic in a given area will always use the auto-away feature. Further, we found that another demographic (which more highly values comfort) will never use it. While this application is simple, it is illustrative in that it shows how our method could be used in the analysis of realistic and potentially more complicated applications.

Our approach has a number of implications for future research. These are discussed below.

Additional Model Checking Analyses

In the presented application, we used formal verification to determine if users with different value functions would use features of the device. However, other analyses are possible. For example, our method could be used to identify design features. Specifically, the default auto-away temperature ($cAutoTemp = 50$) can impact how comfortable people will be when they get home and thus influence their use of auto-away. Analysts could use model checking to discover what auto-away temperature would ensure that all of the considered user types will always employ it. Our method could also be used to consider how different environments will impact feature utilization. For example, the range of possible outside temperatures (*iEnvironmentTemp*) could be changed to reflect different regions in the country. Conversely, the function for computing the cost associated with heating the house could be modified to reflect different types and sizes of house. In both situations, this method could be used to determine how the different user types would use features in the different conditions. Future work should further explore the analyses capabilities of our method.

Other Models for Human Judgment and Decision-Making

Utility theory was appropriate for modeling human decision-making in the presented application due to its economic nature. However, not all human decision-making is appropriate for utility theory (Gintis, 2014). For example, humans may make decisions based on judgments derived from environmental cues or how much they trust automation based on its history of behavior. Such scenarios would be better represented using judgment analysis techniques such as the lens model (Cooksey, 1996) or predictive models of human operator trust (Lee & Moray, 1992) (respectively). However, the extension of EOFM presented here should allow different types of functional models to be accounted for in EOFM tasks. This would allow analysts to model human decision making in different contexts, using the appropriate model, and evaluate the impact these could have on system safety and task performance. Future work should explore how these and other human performance models can be incorporated into EOFM.

Automated Test Case Generation

Model checking can be used for purposes beyond verification. In particular, it can assist in automated test case generation (Ammann & Offutt, 2016). This is a formal process that uses the ability of model checkers to produce traces to create tests that can be executed on an actual system implementation. The nature of this process is such that the tests produced can be guaranteed to see all of the conditions an analyst specifies in coverage criteria. Such tests have two purposes. First, they can be used to validate that a system implementation is consistent with the formal model and thus all of the properties verified against the model are true in the system. Second, analysts can collect metrics during the execution of the test that were not possible to represent in the formal model. For example, a test generated for a human interactive system could be run with real people who can subjectively assess the usability of the system. In previous work (Li & Bolton, ND) we introduced a formal method for generating test using EOFM tasks and formal system models. This approach could be used synergistically with the modeling techniques discussed here to generate complete test sequences for different classes of users. This would give analysts an unprecedented ability to run tests for evaluating the experiences of different users. Future work should research such an approach.

REFERENCES

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

Ammann, P., & Offutt, J. (2016). *Introduction to software testing*. Cambridge University Press.

Bastide, R., & Basnyat, S. (2007). Error patterns: Systematic investigation of deviations in task models. In *Task models and diagrams for users interface design* (pp. 109–121). Berlin: Springer.

Bolton, M. L. (2015). Model checking human–human communication protocols using task models and miscommunication generation. *Journal of Aerospace Information Systems*, *12*(7), 476–489.

Bolton, M. L. (2017a). Novel developments in formal methods for human factors engineering. In *Proceedings of the human factors and ergonomics society annual meeting* (pp. 715–717). Atlanta: Sage.

Bolton, M. L. (2017b). A task-based taxonomy of erroneous human behavior. *International Journal of Human-Computer Studies*, *108*, 105–121.

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

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 Transactions on Systems, Man and Cybernetics: Systems*, *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 B. Weyers, J. Bowen, A. Dix, & P. Palanque (Eds.), *The handbook of formal methods in human-computer interaction* (pp. 343–377). Cham: Springer International.

Bolton, M. L., Bass, E. J., & Siminiceanu, R. I. (2012). Generating phenotypical erroneous human behavior to evaluate human-automation interaction using model checking. *International Journal of Human-Computer Studies*, *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 Transactions on Systems, Man and Cybernetics: Systems*, *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 Transactions on Systems, Man, and Cybernetics, Part A*, *41*(5), 961–976.

Bolton, M. L., Zheng, X., Molinaro, K., Houser, A., & Li, M. (2017). Improving the scalability of formal human-automation interaction verification analyses that use task-analytic models. *Innovations in Systems and Software Engineering*, *13*(1), 1–17.

Calinescu, R., & Kwiatkowska, M. (2009). Using quantitative analysis to implement autonomic it systems. In *Software engineering, 2009. icse 2009. ieee 31st international conference on* (pp. 100–110).

Cámara, J., Garlan, D., Schmerl, B., & Pandey, A. (2015). Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In *Proceedings of the 30th annual ACM symposium on applied computing* (pp. 428–435).

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 Transactions on Systems, Man and Cybernetics Part A: Systems and Humans*, *25*(7), 1054–1075.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model checking*. Cambridge: MIT Press.

Cooksey, R. W. (1996). *Judgment analysis: Theory, methods, and applications.* Academic Press.

de Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M., & Tiwari, A. (2004). SAL 2. In *Proceedings of the 16th international conference on computer aided verification* (pp. 496–500). Springer.

de Moura, L., Owre, S., & Shankar, N. (2003). *The SAL language manual* (Tech. Rep. No. CSL-01-01). Menlo Park: Computer Science Laboratory, SRI International.

Emerson, E. A. (1990). Temporal and modal logic. In J. van Leeuwen, A. R. Meyer, M. Nivat, M. Paterson, & D. Perrin (Eds.), *Handbook of theoretical computer science* (pp. 995–1072). Cambridge: MIT Press.

Fields, R. E. (2001). *Analysis of erroneous actions in the design of critical systems* (Unpublished doctoral dissertation). University of York, York.

Gintis, H. (2014). *The bounds of reason: Game theory and the unification of the behavioral sciences*. Princeton University Press.

Grumberg, O., & Veith, H. (2008). *25 years of model checking: History, achievements, perspectives.* Berlin: Springer.

ISO. (2005). *Ergonomics of the thermal environment – analytical determination and interpretation of thermal comfort using calculation of the pmv and ppd indices and local thermal comfort criteria (ISO standard no.).*

Kirwan, B., & Ainsworth, L. K. (1992). *A guide to task analysis*. London: Taylor and Francis.

Lecerof, A., & Paternò, F. (1998). Automatic support for usability evaluation. *IEEE Transactions on Software Engineering*, *24*(10), 863–888.

Lee, J., & Moray, N. (1992). Trust, control strategies and allocation of function in human-machine systems. *Ergonomics*, *35*(10), 1243–1270.

Li, M., & Bolton, M. L. (ND). Automated test case generation for human-machine interaction.
(Under Review)

Li, M., Wei, J., Zheng, X., & Bolton, M. L. (2017). A formal machine learning approach to generating human-machine interfaces from task models. *IEEE Transactions on Human-Machine Systems*, *47*, 822–833.

Lomuscio, A., Strulo, B., Walker, N. G., & Wu, P. (2010). Model checking optimisation based congestion control algorithms. *Fundamenta Informaticae*, *102*(1), 77–96.

Pan, D., & Bolton, M. L. (2018). Properties for formally assessing the performance level of human-human collaborative procedures with miscommunications and erroneous human behavior. *International Journal of Industrial Ergonomics*, *63*, 75-88.

Paternò, F. (2000). Model-based design of interactive applications. *Intelligence*, *11*(4), 26–38.

Paternò, F., & Santoro, C. (2001). Integrating model checking and HCI tools to help designers verify user interface properties. In *Proceedings of the 7th International Workshop on the Design, Specification, and Verification of Interactive Systems* (pp. 135–150). Berlin: Springer.

Sykes, D., Heaven, W., Magee, J., & Kramer, J. (2010). Exploiting non-functional preferences in architectural adaptation for self-managed systems. In *Proceedings of the 2010 acm ....*

Von Neumann, J., & Morgenstern, O. (1953). *Theory of games and economic behavior*. Princeton University Press.

Weyers, B., Bowen, J., Dix, A., & Palanque, P. (Eds.). (2017). *The handbook of formal methods in human-computer interaction*. Berlin: Springer.

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