Learning Formal Human-machine Interface Designs From Task Analytic Models

Meng Li, Kylie Molinaro, and Matthew L. Bolton
University at Buffalo, State University of New York

User-centered design (UCD) is an approach for creating human-machine interfaces so that they support human operator tasks. UCD can be challenging because designers can fail to account for human-machine interactions that occur due to the inherent concurrence between the human and the other elements of the system. Formal methods are tools that enable analysts to consider all of the possible system interactions using a combination of formal modeling, specification, and proof-based verification. However, the creation of formal models of interface designs can be extremely difficult. This work describes a method that supports UCD by automatically generating formal designs of human-machine interface behavior from task analytic models, where the resulting interface will always support the behavior captured in the task model. This paper describes the method and demonstrates its capabilities with a vending machine application. Results and future research directions are discussed.

INTRODUCTION

User-centered design (UCD) is an approach for creating human-machine interfaces that will support the human operator's tasks (DIS, ISO, 2009). In human factors engineering, task models, the product of task analyses, represent how humans normatively achieve system goals (Kirwan & Ainsworth, 1992). UCDs can be difficult to realize because the inherent complexity of human-machine interaction can result in designers not accounting for human interactions in all situations. Such oversights can result in poor system adoption, decreased productivity, and/or unsafe operations.

Formal methods are tools and techniques that allow analysts to use proof-based techniques to exhaustively consider the different possible system interactions (Clarke, Grumberg, & Peled, 1999). Emerging approaches use formal methods in the design and analysis of human-machine systems (Bolton, Bass, & Siminiceanu, 2013; Degani & Heymann, 2002). Such methods are powerful, but require formal modeling of human-machine interfaces, a process that can be difficult and prone to error (Heitmeyer, 1998). Thus, there is a need for techniques to allow designers to easily create formal interface designs that support operator tasks.

In this paper, we present a method (originally proposed in Bolton and Ebrahimi 2014) that can automatically generate formal models of human-machine interface behavior from task analytic models such that these resulting formal designs will be guaranteed to support the behavior captured by the task models. To do this, we make use of an L* learning algorithm (Angluin, 1987) for learning formal system models.

This paper describes the necessary background for understanding our method, the objectives for its development, and its implementation. We illustrate the capabilities of the method by showing how it can be used to automatically generate the interface for a soda vending machine. Finally, we discuss our results and explore avenues of future research.

BACKGROUND

Formal Methods

Formal methods are tools and techniques for the formal modeling, specification, and verification of systems (Clarke et al., 1999). The formal model describes the behavior of a target system mathematically. Specification properties mathematically describe desirable system conditions (usually using a temporal logic). Formal verification is the process of proving that the system model adheres to the specification. Model checking is a common form of formal verification that performs its proofs automatically using extremely efficient search algorithms (Clarke et al., 1999). In model checking, if a specification property holds over a formal model, the model checker returns a confirmation. If the property is false, the model checker returns a trace through the model, called a counterexample, that shows exactly how the violation occurred. While they are more typically used in the analyses of computer software and hardware, a growing body of work is investigating how formal methods can be used in the engineering of human-machine systems (Bolton et al., 2013).

Formal Models of Human-Machine Interfaces

To be used in formal verifications, human-machine interfaces must be formally modeled. While there are a number of techniques for accomplishing this (see Bolton et al. 2013), all generally follow the tradition set by Parnas (1969), where the interface is a finite state automaton (FSA). In particular, most human-machine interface models are represented as variants of Mealy or Moore machines, where the interface transitions between states are based on human actions or other system events and outputs are either determined by the current state (as with Moore machines; Moore 1956) or by transitions between states (as with Mealy machines; Mealy 1955). In this work, we will use Mealy and Moore machines to formally describe human-machine interface behavior.

Formal Models of Human Task Behavior

Formal models can also be used to represent human tasks in formal verification analyses. Task models can be constructed natively in a formalism or translated into one from a more standard task modeling notation. Formal task models can be paired with formal models of other system behavior (including interfaces) and formal verification analyses can determine if the system model is safe, free from deadlock, or exhibits other desirable usability properties (Bolton et al., 2013).

In the presented work, we are using the Enhance Operator Function Model (EOFM) (Bolton, Siminiceanu, & Bass, 2011). EOFM is an XML-based task modeling language designed to include task analytic human behavior in formal analyses. EOFMs are represented as a hierarchy of goal driven activities that decompose into lower level activities, and at the bottom of the hierarchy, atomic actions. Decomposition operators specify the temporal and cardinal relationships between activities or actions in a decomposition. EOFMs also have conditions on activities that can assert what must be true before an activity can execute (preconditions), when it can repeat (repeat conditions), and what must be true when it finishes (completion conditions). EOFMs have formal semantics that precisely describe how it executes and enables its use in formal verification analyses. Finally, EOFMs have a visual formalism that represents each task as a tree-like graph (examples can be seen later in Figures 3-5). EOFM is used in the work presented here.

L* Learning

L* machine learning is a process capable of generating a formal model based on a series of queries to a teacher oracle (Angluin, 1987). An L* algorithm will learn a minimal FSA for accepting a language (the traditional role of an FSA). It does this by iteratively generating and receiving answers to queries: whether or not specific strings are in the language recognized by the FSA, and whether a given FSA properly recognizes the language. Variants of the original L* algorithm have been developed that allow for different types of FSA to be learned. In particular Raffelt, Steffen, and Berg (2005) have developed LearnLib, a Java-based library that allows for the learning of Mealy machines. In this implementation, the L* algorithm generates queries representing sequences of inputs to the machine. The teacher oracle examines this and returns a sequence of outputs representing the proper machine response. This algorithm is capable of learning models consistent with Mealy machines, thus enabling the automatic generation of human-machine interfaces in the presented work.

Interface Design Generation

Prior work has investigated how to generate human-computer interfaces from task models (García, Lemaigre, González-Calleros, & Vanderdonckt, 2008; Tran, Kolp, Vanderdonckt, Wautelet, & Faulkner, 2010). However, these efforts were concerned with implementing human-computer interfaces, not providing performance guarantees. The work of Combéfis, Giannakopoulou, Pecheur, and Feary (2011) used L* learning to generate interface designs from models of automation behavior so that mode confusion are avoided. Their effort demonstrates that L* learning can generate interfaces that adhere to certain performance properties. However, their approach does not fully consider the human operator task and thus does not facilitate UCD.

OBJECTIVE

In this paper, we describe a method we developed that supports UCD by automatically generating human-machine interfaces from task models that were created as part of user-centered task analyses. In this method, we use human task behavior models represented in the EOFM and the Mealy machine learning capabilities of LearnLib's L* algorithm implementations. Given the properties of the L* learning algorithm and the formal nature of EOFM task models, the resulting learned designs will be guaranteed to always support the human operator's task and thus UCD. In the remainder of this paper, we describe our method, show how it can be used to generate an interface design for a realistic system, discuss our results, and explore future research directions.

METHODS

Our method (Figure 1) takes a task model as input. It extracts two "alphabets": an input one representing the human actions that an interface can receive and an output one representing the information output state of the interface (input variable values to the task model). These are sent to the learner. The learner uses an L* algorithm that creates Mealy machines. It does this through a series of queries to a teacher oracle. The queries contain input sequences from the input alphabet (a series of human actions). The oracle answers the queries by returning corresponding sequences of interface outputs.

The oracle works by first creating a formal model of the human task behavior using a translator. The oracle then uses a model checker to generate valid, query-based traces through this representation to extract the requisite output response. To accomplish this, the formal task behavior representation is paired with a dummy interface model (also generated by the translator) that is only capable of initializing interface outputs and allowing their values to change in response to human actions (see the formal model architecture in Figure 2). Further, the transition logic in the dummy interface model ensures that each interface output can only change if the action being executed is part of a valid task execution sequence. To ensure that the model will only consider the input/action sequence contained in a query, another formal model is also included. This model uses a synchronous observer architecture (Rushby, 2012) to track the sequence of human actions that have occurred. If the sequence from the query has been observed, an indicator variable (*SequenceObserved*; Figure 1) becomes and stays true.

The aggregate formal model is then employed by the teacher oracle that uses a model checker to prove the temporal logic property that the indicator variable will never be true $[\mathbf{G}\neg(\mathit{SequenceObserved})]$. The effect of this is that, if the human action sequence in the query is valid, the model checker will return a counterexample showing how the sequence can occur. The oracle extracts the output alphabet sequence corresponding to the input query from the counterexample. This is sent back to the learner as a response.

Through iterative queries and responses between the learner and the oracle, the learner will ultimately learn a Mealy machine representation of the human-machine interface. In this, the inputs represent human actions, outputs represent the state of interface display information, and state represents the internal state of the interface.

The method was implemented in Java using EOFM task models, the symbolic analysis laboratory's (SAL's) model checkers (De Moura, Owre, & Shankar, 2003), the EOFM-to-SAL translator (Bolton et al., 2011), and LearnLib's (Raffelt et al., 2005) L*-based Mealy machine learning algorithm.

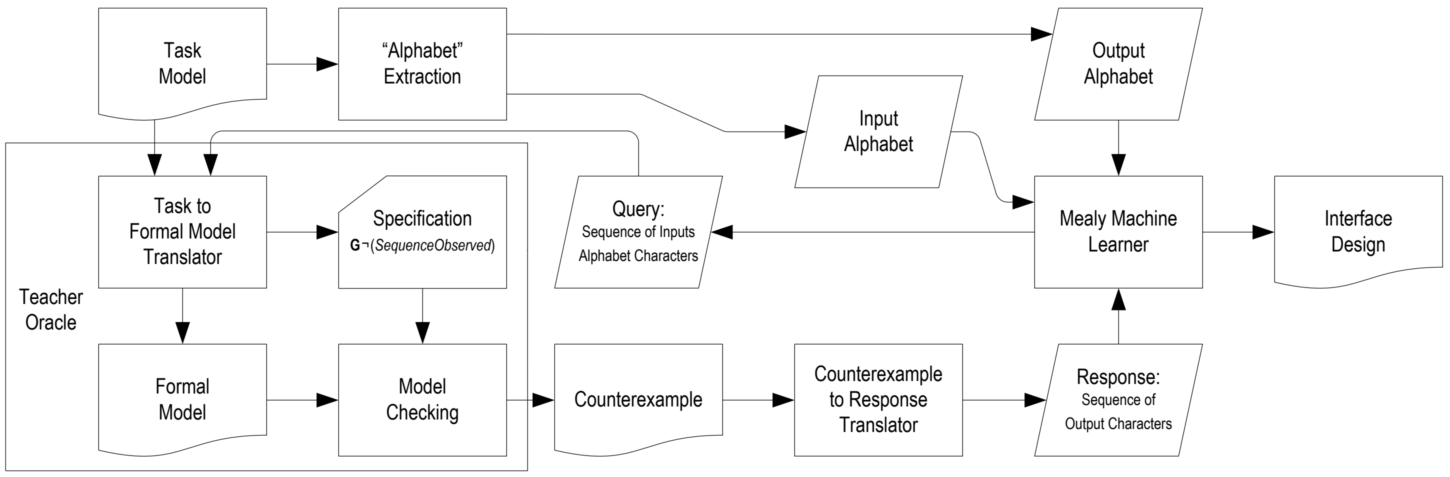
Task Model
"Alphabet" Extraction
Output Alphabet
Input Alphabet
Task to Formal Model Translator
Specification G¬(SequenceObserved)
Query: Sequence of Inputs Alphabet Characters
Mealy Machine Learner
Interface Design
Teacher Oracle
Formal Model
Model Checking
Counterexample
Counterexample to Response Translator
Response: Sequence of Output Characters

Figure 1. The method for automatically generating human-machine interfaces from task models and usability properties. Details of the formal model's architecture can be seen in Figure 2.

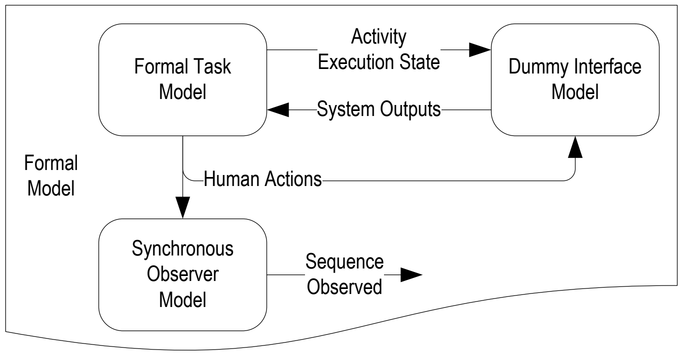

Figure 2. Formal model architecture used by the teacher oracle (Figure 1).

APPLICATION

To demonstrate the capabilities of our method, we use it to learn the interface of a simplified beverage vending machine. For this machine, we assume that it is only able to sell only one kind of drink, where drinks cost 50 cents and the machine exclusively accepts payment with quarters.

Task Modeling

We created an EOFM task model describing how we would normatively want the human operator to interact with the machine (Figures 3–5). This model assumes that the human operator can see how much money has been entered into the machine (iMoneyIn), if a drink has been vended (iDrinkOut), and if any change has been returned (iMoneyOut). He or she can perform actions for entering quarters (hEnterQuarter), pressing the drink button (hPressDrinkButton), pressing the change return (hPressChangeReturn), picking up vended drinks (hPickUpDrink), and picking up returned change (hPickUpChange). The behavior of the human operator was described using three goal directed tasks for: entering money, acquiring a drink, and retrieving change. Each task is described below.

Figure 3 shows the task for entering money. This task can be performed when the money entered is less than the drink price. To enter money, the human first notes how much money is currently in the machine (lChangeIn = iMoneyIn from aRememberChangeInValue). Then, the human operator performs the activity for entering a quarter (aEnterCoin). This is completed when the hEnterQuarter action is performed and the completion condition, that the money entered the machine is now a quarter's value more than before the new quarter was entered (iMoneyIn = lChangeIn + cQuarter), is satisfied.

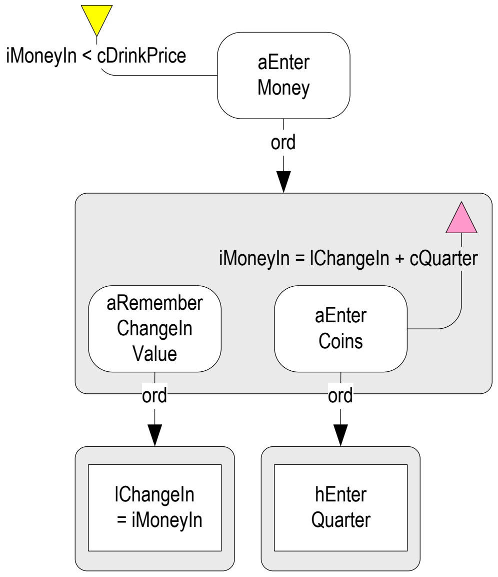

Figure 3. Visualization of the EOFM task for entering money into the drink vending machine. Activities are rounded rectangles, actions are unrounded rectangles. Preconditions and completion conditions are yellow and magenta triangles, respectively, connected to their associated activities and annotated in condition logic. Activity decompositions are represented as downward pointing arrows annotated with a decomposition operator. Only the ordered (*ord*) decomposition is used here.

The task for getting a drink is shown in Figure 4. In this task, once the money entered is greater than or equal to the drink price, the human operator first presses the drink button, which should result in a drink output and the entered money resetting to zero. The human operator can then pick up the drink, which should result in there no longer being a drink output.

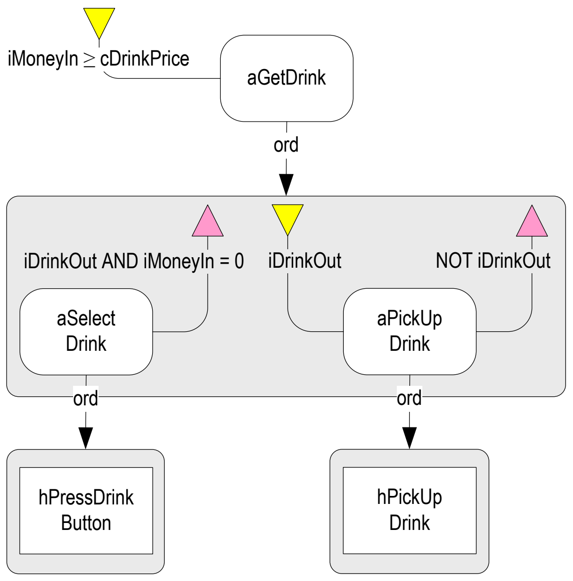

Figure 4. Visualization of the EOFM task for dispensing and picking up a drink from the machine.

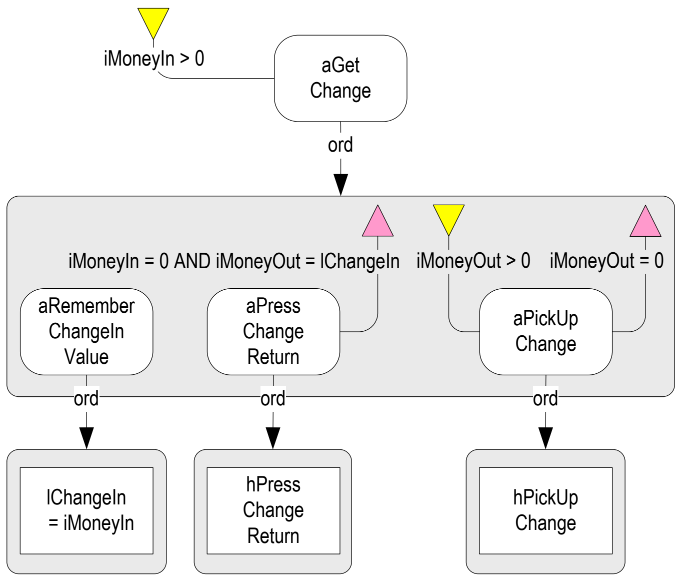

Figure 5. Visualization of the EOFM task for picking up change returned by the machine.

If the money entered in the machine is greater than zero, the human operator can perform the task for returning change (Figure 5). He or she first notes how much money is currently in the machine (as was done when entering money) and then presses the change return button. For this to complete successfully, the money in the machine must then drop to zero and the money outputted must match what was in the machine before the change return was pressed.

Interface Generation

This task model was used as input to the Java program implementation of our method. The program ran for 124.41 seconds during which 186 queries were processed. The resulting interface model was converted from a Mealy machine to a Moore machine (Figure 6) to improve readability.

Results

An examination of the generated interface (Figure 6) reveals that it meets the goals of the generation method: creating an interface design that will always be compatible with the human operator task. The machine allows the human operator to enter money until the correct drink price is reached. The machine will only vend a drink if the entered money is equal to the drink price. If a drink is output, then the human operator can pick it up, removing it from the machine. Whenever change has been entered, the human can press the change return to get the money as output, after which they can pick up the money.

An interesting feature of this interface design is that it uses forcing functions to keep the human operator on task. For example, once the money entered in the machine matches the drink price, the machine will no longer accept quarters as input. This behavior is different from drink machines most people are familiar with. This is discussed in the next section.

DISCUSSION AND FUTURE WORK

The work presented in this paper described a novel approach for using L* learning to automatically generate human-machine interfaces. By learning interfaces from task analytic behavior models, this method ensures that the interface will always support the human operator's task behavior as captured in by a task analysis and thus supports UCD. Further, because L* learning ensures that a minimal model is produced, the statespace of the interface will also be minimal. This is a potentially advantageous property because it ensures the minimal complexity of the interface (Combéfis et al., 2011). The fact that the generation process produced forcing function behavior in the generated human-computer interface represents an interesting phenomena that may or may not support good usability. This and other factors will be explored in extensions of this effort.

Formal Verification of Generated Interfaces

While the interface generated in the presented application was easy to examine manually, more complex applications may not afford such straightforward assessments. Patterns of formal specifications properties exist for checking that interfaces support usability (Bolton et al., 2013) and human operator tasks (Bolton, Jimenez, van Paassen, & Trujillo, 2014). Future work will investigate which of these properties are supported by the presented generation method.

Additional Applications

The application presented in this paper is illustrative, but simple. Additional complications could be discovered as the method is used to generate more complex interfaces. For example, because the method uses model checking, it is subject to model checking's scalability limitations (Clarke et al., 1999). Thus, as we use the method to generate interfaces for more complex applications, we will evaluate how the method scales.

Task Modeling for Interface Generation

In creating the task analytic models for the presented application, several features of task models revealed themselves to be important for successfully generating interfaces. First, it was best to keep the task models as modular as possible. This

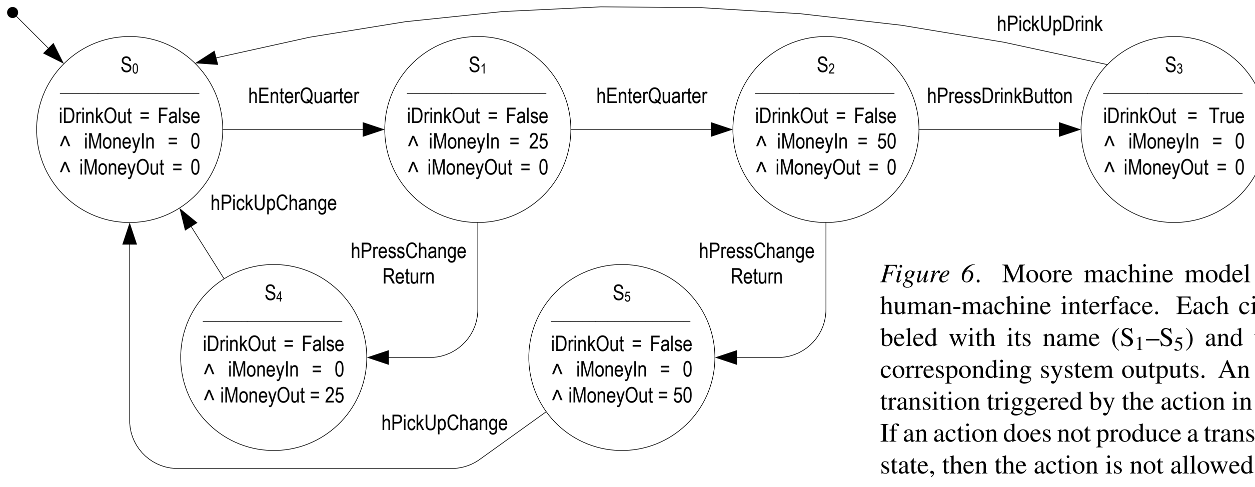

Figure 6. Moore machine model of the generated human-machine interface. Each circle is a state labeled with its name (S_1–S_5) and the values of the corresponding system outputs. An arrow indicates a transition triggered by the action in the arrow's label. If an action does not produce a transition from a given state, then the action is not allowed in that state.

ensured that model checking process could successfully find desired execution sequences. Second, the task models needed to be explicit about the conditions expected from the interface. For example, when inserting money, the task model was required to note the amount of money entered into the machine before inserting an additional quarter. Further, the task's completion condition had to assert that the amount of money registered in the machine was increased by 25 cents. Third, while non-determinism is supported in EOFM (Bolton et al., 2011), the L* learning algorithm assumes consistency in the answers to its queries. This could result in situations where the L* learning algorithm will fail to produce an interface. Thus, it is best to keep task models deterministic. It is not clear if these modeling practices will prove themselves to be compatible with standard task modeling process. Future work will investigate this.

Incorporation of Usability Principles

Beyond task related properties, there are a variety of formal usability specification properties that have been identified (Bolton et al., 2013). Thus, it should be possible to incorporate usability properties into the generation process discussed here. This will be the subject of future research.

Finally, the work discussed here did not consider the graphical representation of the human-machine interface. However, the action flow information contained in a task model could conceivably be used to influence the position and layout of interface controls. Future work will investigate whether this can be accounted for in the interface generation process.

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1429910.

REFERENCES

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and computation*, *75*(2), 87–106.

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*, 488–503.

Bolton, M. L., & Ebrahimi, S. (2014). An approach to generating human-computer interfaces from task models. In *2014 aaai spring symposium series* (pp. 92–97). Palo Alto: AAAI.

Bolton, M. L., Jimenez, N., van Paassen, M. M., & Trujillo, M. (2014). Automatically generating specification properties from task models for the formal verification of human-automation interaction. *Human-Machine Systems, IEEE Transactions on*, *44*(5), 561–575.

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.

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

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

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

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.

DIS, ISO. (2009). 9241-210: 2010. Ergonomics of human system interaction-part 210: Human-centred design for interactive systems. *International Organization for Standardization (ISO), Switzerland*.

García, J. G., Lemaigre, C., González-Calleros, J. M., & Vanderdonckt, J. (2008). Model-driven approach to design user interfaces for workflow information systems. *Journal of Universal Computer Science*, *14*(19), 3160–3173.

Heitmeyer, C. (1998). On the need for practical formal methods. In *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time Fault-Tolerant Systems* (pp. 18–26). London: Springer.

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

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal*, *34*(5), 1045–1079.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata studies*, *34*, 129–153.

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

Raffelt, H., Steffen, B., & Berg, T. (2005). Learnlib: A library for automata learning and experimentation. In *Proceedings of the 10th international workshop on formal methods for industrial critical systems* (pp. 62–71).

Rushby, J. (2012). The versatile synchronous observer. In R. Gheyi & D. Naumann (Eds.), *Formal methods: Foundations and applications* (Vol. 7498). Springer Berlin Heidelberg.

Tran, V., Kolp, M., Vanderdonckt, J., Wautelet, Y., & Faulkner, S. (2010). Agent-based user interface generation from combined task, context and domain models. In *Proceedings of the 8th international workshop on task models and diagrams for user interface design* (pp. 146–161). Berlin: Springer.