# A Formal Machine–Learning Approach to Generating Human–Machine Interfaces From Task Models

Meng Li, Jiajun Wei, Xi Zheng, and Matthew L. Bolton, *Senior Member, IEEE*

***Abstract*—User-centered design (UCD) is an approach for creating human–machine interfaces that are usable and support the human operator's tasks. UCD can be challenging because designers can fail to account for human–machine interactions that occur due to the concurrency between the human and the other system elements. 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, creating formal interface design models 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. The resulting interface design will always support the behavior captured in the task model. This paper describes the method and demonstrates its capabilities with three case studies: a light switch, a vending machine, and a patient-controlled analgesia pump. The produced designs are validated with formal verifications to prove that they support their associated tasks. Results and future research are discussed.**

***Index Terms*—Formal methods, human–automation interaction, machine learning, task analysis, user-centered design (UCD).**

## I. Introduction

USER-CENTERED design (UCD) is an approach for creating human–machine interfaces to usably support human operator tasks [2]. UCD can be difficult because the inherent complexity of human–machine interaction can result in designers not accounting for interactions in all situations. Such oversights can result in poor system adoption, decreased productivity, and unsafe operations.

Formal methods are tools and techniques that allow analysts to use proof-based techniques to exhaustively consider the different possible system interactions [3]. Emerging approaches use formal methods in the design and analysis of human–machine systems [4]–[6]. Such methods are powerful, but require formal modeling of human–machine interfaces, a process that can be difficult and prone to error [7]. Thus, there is a need for techniques to allow designers to easily create formal interface designs that support operator tasks.

We present a method (originally proposed in [1] and [8]) that can automatically generate formal models of human–machine interface behavior from task-analytic models, products of task analyses used to represent how humans normatively achieve system goals [9], [10]. The resulting formal designs will be guaranteed to support the behavior represented in the task models. To do this, we make use of an L* algorithm [11] 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 using it to automatically generate interfaces from task models for several case studies. Furthermore, we systematically validate the interface designs that result from our method by formally verifying task-based usability properties against them. We ultimately discuss our results and explore avenues of future research.

## II. Background

### A. Formal Methods

Formal methods are tools and techniques for the modeling, specification, and verification of systems [12]. A formal model describes the behavior of a target system mathematically (usually as a finite-state transition system). 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. It performs its proofs automatically using extremely efficient search algorithms [3]. In model checking, if a specification property holds for 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. This shows exactly how the violation occurred. While they are predominately used in the analyses of computer hardware and software, a growing body of work is investigating how formal methods can be used in the engineering of human–machine systems [4].

### B. 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

Manuscript received August 23, 2016; revised December 3, 2016 and March 19, 2017; accepted April 8, 2017. This work was supported by the National Science Foundation under Grant IIS-1429910. A subset of the results described in this paper was presented at the 2015 Human Factors and Ergonomics Society (HFES) Annual Meeting, Los Angeles, CA, USA, October 26–30 [1]. This paper constitutes a substantial contribution beyond what is reported in the HFES paper. Specifically, the version of the method presented here operates differently from the one in [1]. Furthermore, we present two additional case studies beyond the vending machine application from [1]. We also present additional verification and validation results and an extended discussion. This paper was recommended by Associate Editor C. Burns. *(Corresponding author: Matthew L. Bolton.)*

The authors are with the Department of Industrial and Systems Engineering, University at Buffalo, The State University of New York, Amherst, NY 14260-2050, USA (e-mail: mli42@buffalo.edu; jwei7@buffalo.edu; xzheng 24@buffalo.edu; mbolton@buffalo.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/THMS.2017.2700630

techniques for accomplishing this (see [4]), all generally represent the interface as a form of finite-state automaton (FSA). Most of the human–machine interface models are represented as variants of Mealy or Moore machines. A Mealy machine [13] represents a system as an FSA, where each transition between states has an associated input (which causes the transition) and output. A Moore machine [14] is also an FSA with both inputs and outputs, where inputs cause transitions between states. However, outputs are associated with each state rather than each transition. Both have equivalent expressive power. In this work, we will use Mealy and Moore machines to formally describe human–machine interface behavior.

### C. *Task-Analytic Behavior Models*

In a cognitive task analysis, an analyst conducts interviews, performs field studies, and inspects training materials to describe how (physically and cognitively) human operators normatively achieve goals with a system [9], [10]. This is usually documented using a hierarchal task model. Such a model is a collection of individual tasks. Each is represented as a hierarchy of goal-directed activities that decompose into other activities and (at the lowest level) atomic actions. Strategic knowledge (condition logic) controls when activities can execute and describes what each activity should accomplish. Modifiers between activities or actions control how they execute in relation to each other.

Task-analytic models are some of the most successful technologies developed by human factors engineers. Task models can be used in interface design [15], training development [16], usability analyses [17], real-time monitoring programs [18], and system safety and performance analyses with both normative and erroneous human behavior [19]–[22]. In fact, task-analytic models are important to UCD [23], a broad approach to developing human–machine interaction to accommodate human task goals.

### D. *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 [20], [24]–[29], free from deadlock [30]–[32], or exhibits other desirable usability properties [33], [34]. See [4] for a deeper review of this literature.

There are a number of different task-analytic modeling formalisms. These include ConcurTaskTrees (CTT) [35], its extension Hamster [36], AMBOSS [37], and the enhanced operator function model (EOFM) [20]. EOFM is expressive, platform independent, and feature rich. It has a formal semantics [20], [38] and has been used in a number of human factors analyses [20]–[22], [25], [34], [39]–[44]. Thus, it will be used for the work discussed in this paper.

EOFM is a formal XML-based task modeling language. It represents human tasks as input–output systems. Input variables represent observable information from external sources (like human–machine interfaces and the environment). Local variables represent human memory and the execution state of the human's task. Outputs are human actions.

EOFMs are represented as a hierarchy of goal-directed activities that decompose into lower level activities and, at the bottom of the hierarchy, atomic actions. 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. Specifically, the execution behavior of a task model is represented by treating each activity and action as an FSA. Each automata transitions between three different execution states (Ready, Executing, and Done) based on the evaluations of Boolean expressions asserted using task input variables, task local variables, and the execution state of the other activities and actions in the task. More details can be found in [20] and [38].

Finally, EOFM tasks can be visualized as tree-like graphs (examples can be seen later in Figs. 3, 4, and 6) [45].

### E. *L* Learning*

L* machine learning is a process capable of generating a formal model based on a series of queries to a teacher oracle [11]. 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 *et al.* [46] 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 these queries 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.

### F. *Interface Design Generation*

Prior work has investigated how to generate human–computer interfaces from task models [47]–[52]. These generators were based on transformation rules, which can be viewed as formal. However, these efforts were concerned with rapid prototyping and shortening development time. As such, they employ heuristics to produce interface descriptions at different levels of abstraction for use in larger interface design and analysis processes. While desirable interface properties can be achieved with these techniques, they do not, nor do they intend to provide formal guarantees.[1]

[1]When transformation rules can ensure that formal specifications are satisfied in the resulting representation, this is called formal refinement. We explore this more thoroughly in Section VIII-C.

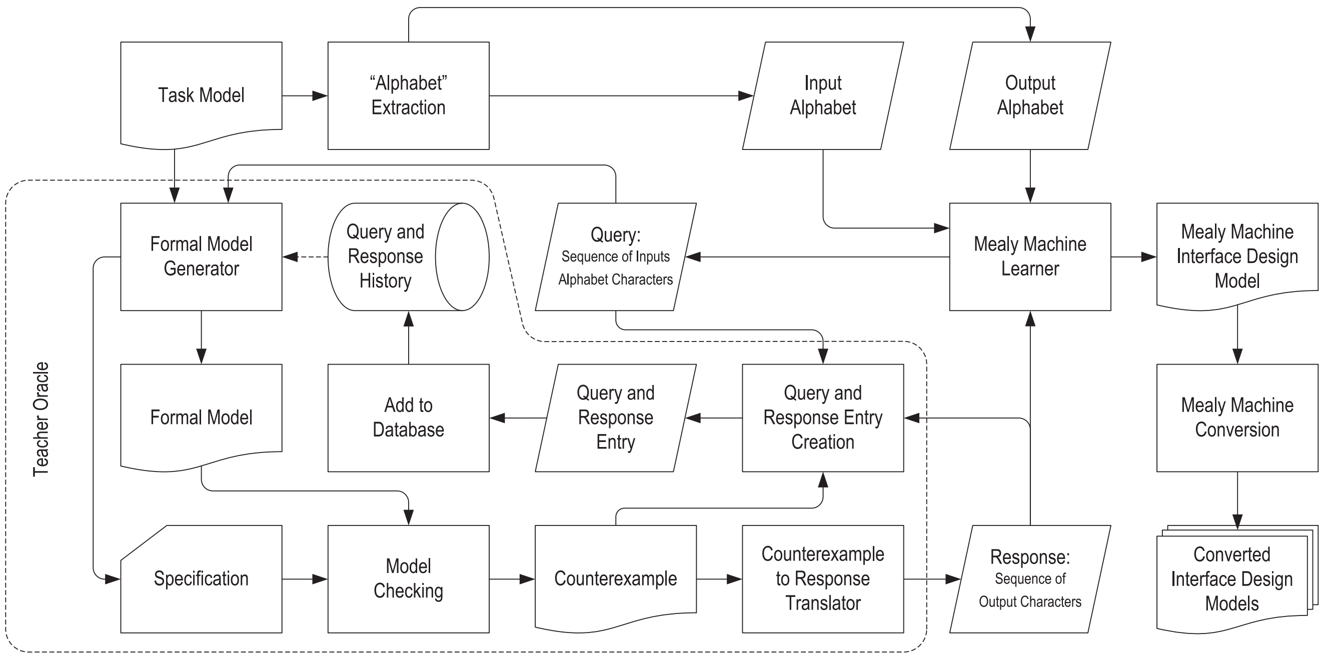

Task Model
"Alphabet" Extraction
Input Alphabet
Output Alphabet
Formal Model Generator
Query and Response History
Query: Sequence of Inputs Alphabet Characters
Mealy Machine Learner
Mealy Machine Interface Design Model
Teacher Oracle
Formal Model
Add to Database
Query and Response Entry
Query and Response Entry Creation
Mealy Machine Conversion
Specification
Model Checking
Counterexample
Counterexample to Response Translator
Response: Sequence of Output Characters
Converted Interface Design Models

Fig. 1. Method for automatically generating human–machine interfaces from task models. Details of the formal model's architecture can be seen in Fig. 2.

Heymann and Degani [53] investigated a formal algorithmic approach for generating interface designs from models of system automation that would avoid mode confusion (a condition where the human is unable to keep track of the system's state or mode [54], [55]). The work of Combéfis *et al.* [56], [57] extended this approach by using L* learning to generate interface designs from models of automation behavior to ensure that the human operator maintains "full control" (a condition that will nominally prevent mode confusion [58]). These projects were both formal and provided safety guarantees, as mode confusion has been implicated in a number of human–automation interaction accidents [54], [59], [60].

Collectively, the work reviewed in this section demonstrates that interfaces can be generated from task models and that L* learning can be used to generate interfaces that adhere to certain performance properties. However, none of these approaches provide guarantees about the performance of the human operator task and thus do not facilitate UCD.

## III. Objective

In this paper, we describe a method that we developed to support UCD by automatically generating human–machine interfaces from task models, where we presume that the task models are created as part of cognitive task analyses. In this method, we use task behavior represented in the EOFM and the Mealy machine learning capabilities of LearnLib's L* algorithms. 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 will be supported. Because this method is generating interfaces from task-analytic models, it is accounting for the high validity such approaches have shown in the extended human factor literature.

In the remainder of this paper, we describe our method, show how it can be used to generate an interface design for several systems, validate our designs with formal verification, discuss our results, and explore future research directions.

## IV. Method

Fig. 1 presents our method. This takes a task model as input and extracts two "alphabets." An input alphabet represents the human actions that an interface can receive. An output alphabet represents the state of the information outputs of the interface: all of the different combinations of task input variable values. The alphabets are sent to a 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 (see Fig. 2) using a translator. This formal model contains a formal task submodel. This has outputs representing the human actions that are performed as well as a Boolean indicator variable (NoTask) that is true when no tasks are executing and false otherwise. The formal task submodel is paired with a dummy interface submodel that is also generated by the translator from the task-analytic behavior model. This dummy interface is only capable of initializing interface outputs and allowing their values to change in response to human actions, where the values assumed by interface outputs can be any possible value of that output variable. To ensure that the model will only consider the input

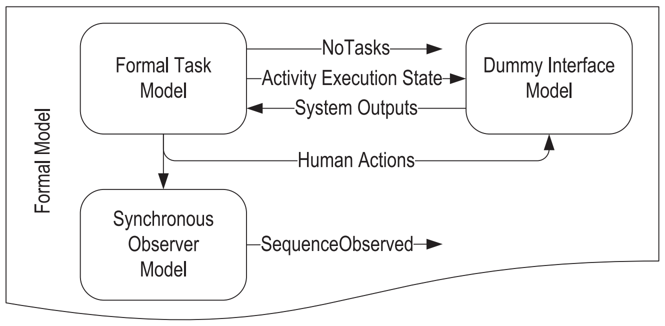


Fig. 2. Formal model architecture used by the teacher oracle (see Fig. 1).

action sequence contained in a query, we use a synchronous observer submodel [61] to track the sequence of human actions that have occurred. If the sequence from the query has been observed, a Boolean indicator variable (SequenceObserved; Fig. 2) becomes and stays true.

The teacher oracle then uses a model checker on the formal model to prove the following temporal logic property:

$$\mathsf{G}\neg(SequenceObserved \wedge NoTasks).$$

This asserts that, in all paths through the model, it should never be true that the action sequence from the learner is observed ($SequenceObserved$) and that all tasks have completed their execution ($NoTasks$). The effect of this is that, if the human action sequence in the query is executable and allows the human to successfully complete his or her tasks, 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.

In an earlier version of the method (see [1]), every instance of the formal model was always initialized to the same initial default state. However, this produced situations where the formal model would take excessive amounts of time to be evaluated. Thus, to improve the scalability of the method, we incorporated a database ("Query and Response History" in Fig. 1) that keeps a history of all of the queries, the output responses, and the state of the models at the end of the queries. Thus, when a new query arrives, the oracle checks to see if the action sequence or any prefix to it is in the database. Note that this search is done backward to ensure that longest preevaluated prefix is selected. If this check returns something, then the formal model is initialized to the state it was in after the execution of the prefix. The synchronous observer is set only to look for the action sequence that followed the prefix in the original query. Once the model-checking process has been completed, the oracle creates a response that is a concatenation of the original output sequence for the prefix and the output sequence seen for the actions that followed the prefix.

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. While the Mealy machine representation is sufficient, we generally found it easier to interpret results when they were represented as a Moore machine [62] (where interface states map to outputs instead of transitions mapping to outputs). Thus, our method allows for this conversion. Furthermore, we needed a formal representation of the model for use in formal verification analyses for validation experiments. To address this, we converted the Moore machine representation into the input language of a model checker.

The method was implemented in Java using EOFM, the Symbolic Analysis Laboratory's (SAL's) model checkers [63],[2] the EOFM-to-SAL "optimized" translator [38], and LearnLib's [46] L*-based Mealy machine learning algorithm.[3]

## V. Applications

To demonstrate the capabilities of our method, we use it to learn the interface for three different applications: a light switch, a beverage vending machine, and a patient-controlled analgesia (PCA) pump. All of the interfaces in the application were learned using our Java-based implementation of the method on a computer workstation with a 3.7-GHz Intel Xeon processor and 128 GB of RAM running Linux Mint.

### A. Light Switch

The first application is a simple light switch. In this, a person flips a single switch to turn a light ON and OFF.

*1) Task Modeling:* The task model for describing the desired user behavior for the light switch is shown in Fig. 3: (a) represents the behavior for turning a light ON when it is OFF and (b) represents the behavior for turning the light OFF when it is ON. The state of the light is represented by a Boolean variable (iLight) that is true when the light is ON and false otherwise. In both cases, the top-level activity is completed by performing the action for flipping the switch (hFlipSwitch).

*2) Interface Generation:* When the task model was run through the interface generation method, the generation process took 2.65 s and processed 15 queries. The generated interface design is shown in Fig. 3(c). This appears to support the task [see Fig. 3(a) and (b)] in that the human flipping the switch will turn the light ON and OFF.

### B. Vending Machine

For the vending machine application, we assume that the machine only sells one kind of drink, drinks cost 50¢, and the machine exclusively accepts payment with quarters.

*1) Task Modeling:* We created an EOFM task model describing how we would normatively want the human operator to interact with the machine (see Fig. 4). This

[2]Note that we did experiment with the use of bounded model checking (an approach that limits model search depth) in our analyses. However, this proved to be computationally less efficient. This is because we could not make any guarantees about the lack of deadlock in generated formal models and thus needed to run multiple bounded model-checking runs to avoid artificial confirmations that would not produce counterexamples.

[3]The method's implementation and model data for all of the presented applications are available at http://fhsl.eng.buffalo.edu/EOFM/

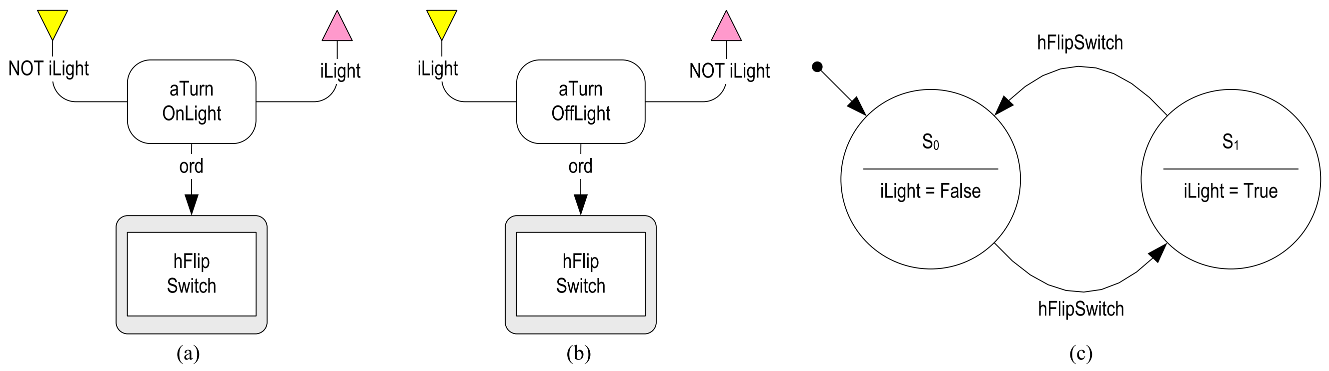


Fig. 3. (a) and (b) Visualization of the EOFM task for turning the light ON and OFF. Activities are rounded rectangles; actions are unrounded rectangles. Preconditions and completion conditions are yellow and magenta triangles, respectively, connected to their associated activities and annotated in condition logic. Activity decompositions are downward pointing arrows annotated with decomposition operators. (c) Moore machine model of the generated human–machine interface. Each circle is a state labeled with its name ($S_0$ and $S_1$) 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.

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.

Fig. 4(a) 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 is satisfied: that the money entered into the machine is a quarter's value more than it was before the quarter was entered (iMoneyIn = lChangeIn + cQuarter).

The task for getting a drink is shown in Fig. 4(b). In this, once the money entered is greater than or equal to the drink price, the human operator first presses the drink button. This 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 being no drink output.

If the money entered into the machine is greater than zero, the human operator can perform the task for returning change [see Fig. 4(c)]. 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.

*2) Interface Generation:* This task model was used as input to the Java program implementation of our method. The program ran for 61.82 s, during which 185 queries were processed. The resulting interface is shown in Fig. 5.

An examination of the generated interface seems to show that the interface is compatible with the associated 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.

## VI. PCA PUMP

Our final application was the interface to a PCA pump. A PCA pump is a medical device designed to deliver pain medication to a patient intravenously based on patient inputs (usually via a button) and a prescription programmed into it by a technician. This application was chosen specifically because PCA pumps have been known to have a number of issues related to human–machine interaction. Furthermore, a formal reference model of a PCA pump and its interface have been developed [64]. Because the paper associated with this model contains a description of a human operator task (though not an explicit task model), this application gives us the opportunity to check our generation method against a robust reference.

The PCA reference model [64] has five major components: alarm manager, pump controller, user interface, display, and logs for events and errors. For this work, we focus on the user interface and the behavior it was intended to support.

The PCA pump interface is designed to give practitioners and patients access to information and controls necessary for setting up the pump and delivering medication. The PCA pump interface is meant to have four states to indicate the state of the pump: Stopped, Started, Bolus, and Wait. The Stopped state represents a situation where the device is not administering any treatment. The Started state indicates that the pump is delivering medication at a steady rate based on a prescription. In the Bolus state, the pump delivers an additional dose of medication. When the pump interface is in the Wait state, it indicates that prescription or bolus delivery has been paused.

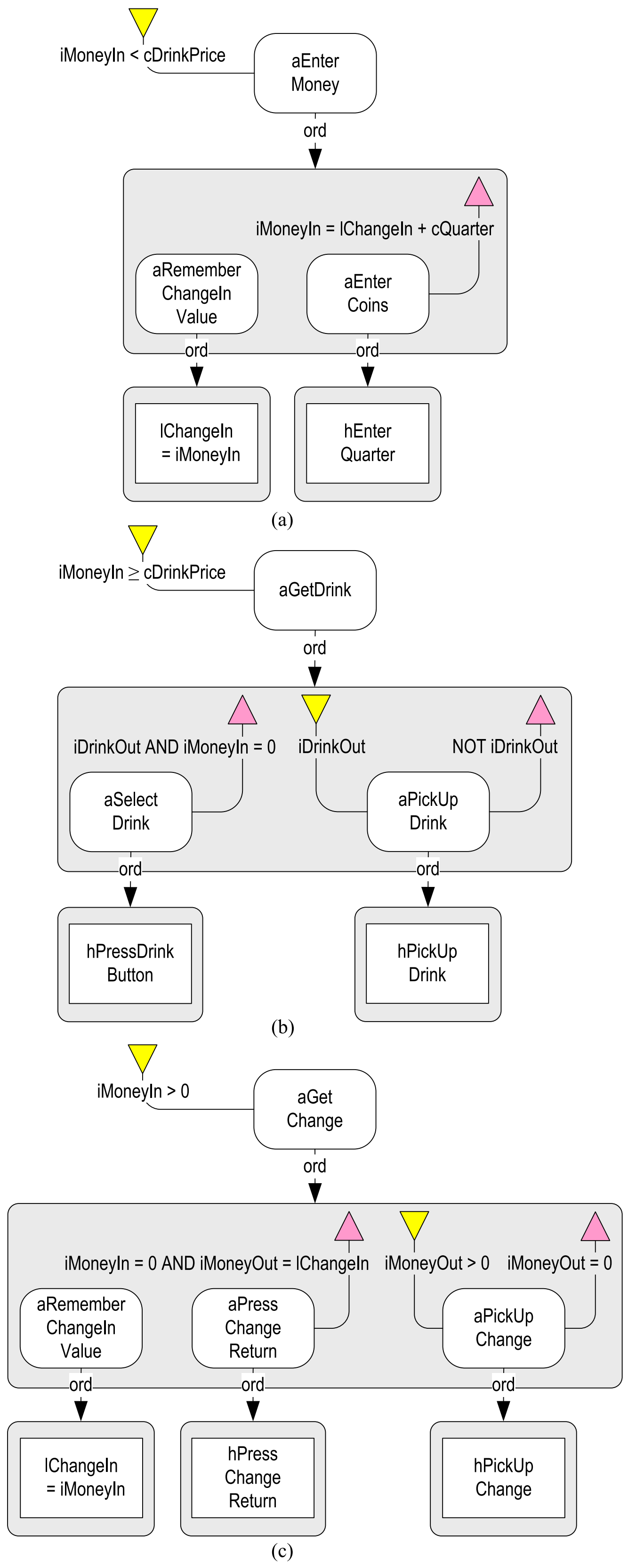


Fig. 4. Visualization of the EOFM tasks for (a) entering money into the drink vending machine, (b) dispensing and picking up a drink from the machine, and (c) picking up change returned by the machine.

The pump interface also keeps track of whether or not a bolus is being administered so that a bolus can resume administration if it is interrupted.

Humans (patients or medical practitioners) who interact with the device can do so in several ways. The interface is meant to allow for all of the following six human actions: a change to the prescription programmed into the device,[4] confirming treatment changes, starting a bolus, canceling a bolus, starting the administration of a prescription, waiting for a bolus to complete, and waiting for treatment to administer.

It is important to note that the interface in the reference model [64] is not necessarily a good example of interface design. It is being considered here because it gives us a robust standard that our method's results can be compared to.

## A. Task Modeling

An EOFM model was instantiated to describe the task behavior from the reference model. In this, the human is able to see the interface state, iState, which can have any of the four values identified above. The human can also see the state of bolus administration (iBolus), which is true during bolus administration. The human operator could perform all of the following actions, each corresponding respectively to the interface actions listed above: hChangeSetting, hConfirmSettings, hPressBolus, hPressStart, hWaitForBolus, and hWaitForInfusion.

Fig. 6 shows the task describing how the human operator interacts with the device. Infusion can be started [see Fig. 6(a)] when the interface is in the Stopped state by performing the hPressStart action. A human can change prescription settings [see Fig. 6(b)] if the interface is in the Started or Bolus states by performing the hChangeSettings action. The human can wait for infusion to finish [see Fig. 6(c)] when the interface is in the Started state by performing the hWaitForInfusion action. A human can start a bolus [see Fig. 6(d)] once infusion has Started by executing the hPressBolus action. Note that this not only changes the interface state, but sets iInBolus to true. A human can wait for the bolus infusion to finish [see Fig. 6(e)] by waiting (hWaitForBolus). Note that when a bolus completes, iInBolus becomes false. If the interface is in the Started or Bolus states, the human can change the prescription [see Fig. 6(f)] in the pump using the (hChangeSettings) action, which puts the interface into the Waiting state. Finally, if the interface is in the waiting state, the human can confirm treatment [see Fig. 6(g)] by performing the settings confirmation action (hConfirmSettings). If bolus treatment is not being administered, this should put the interface back into the Started state. Otherwise, it should put it into the Bolus state.

## B. Interface Generation

When the PCA task model was run through our implementation of the method, it produced the interface shown in Fig. 7(a) after 87.84 s and 217 queries. As with the previous applications, a visual inspection appears to show that the interface is consistent with the task used to generate it.

[4]This represented as a single action in the reference model interface [64].

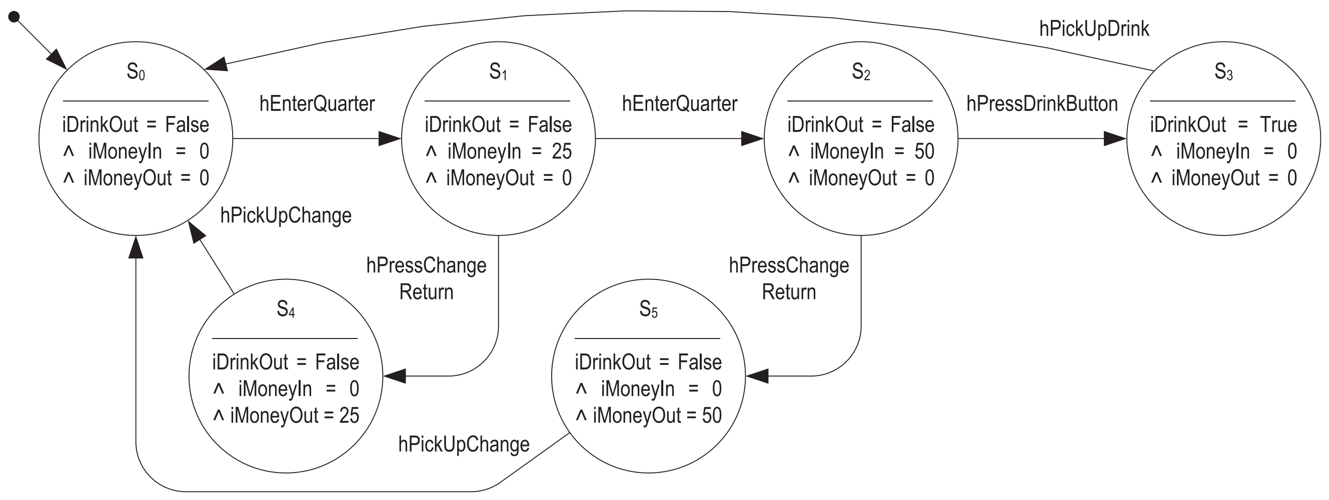


Fig. 5. Moore machine model of the generated vending machine human–machine interface.

Because the PCA application comes from a reference model [64], there is an established interface model to compare the generated model to. This is shown in Fig. 7(b). While the interface we generated contains five states, the reference model contains only four. However, the reference model uses a notation that is a combination of Mealy and Moore machine features. Specifically, transitions with the expression $\mathsf{inBolus} := \mathsf{true}$ allow variables to be assigned values when a transition occurs. This allows the interface to be in the Wait state with inBolus being both true and false. As such, we can think of the Wait state from Fig. 7(b) being a combination of states $S_3$ and $S_4$ from Fig. 7(a). With this perspective, Fig. 7(a) and (b) clearly represent the same interface behavior. Thus, our interface generation method was able to create the same design that was included with and rigorously evaluated in the PCA formal reference model.

## VII. Validation

The visual inspection of the generated interfaces seems to indicate that they always support the tasks they were generated from. However, to validate that this is the case, we can use formal verification. Specifically, EOFM supports the ability to automatically generate specification properties to check that a system and/or interface is compatible with and always supports the human operator task [40].

Generated properties assert qualities about the execution state of the task models based on their formal semantics [40]. This allows analysts to look for problems in the human-automation interaction (HAI) of a system by finding places, where the task models would not perform as expected and thus elucidate potential unanticipated interaction issues. Properties are able to assert that every execution state of each activity and action are reachable, that every transition between execution states are achievable, that all activities and actions will eventually finish, and that any human task will always eventually be performable. Collectively, if all of these properties evaluate to true, then we know that the system being evaluated always supports the task.

We used this feature of EOFM to validate each of the interfaces generated above. Specifically, each formal interface model was paired with the formal version of the EOFM task model used to generate it and checked against all of the generated specification properties supported by EOFM [40]. Additionally, because some of these properties are only true if the model has no deadlock states, formal deadlock checking was also performed. All of these verification analyses and deadlock checks were performed using SAL on the same machine used to generate the interfaces.

TABLE I
Generated Interface Formal Verification Results

| Model States | Interface States | Model Properties | Verified Time (s) | Verification Time (s) | Deadlock |
|---|---|---|---|---|---|
| Light Switch | 2 | 40 | 27 | 0.83 | 0.05 |
| Vending Machine | 6 | 475 | 104 | 41.95 | 0.57 |
| PCA Pump | 4 | 7808 | 111 | 36.64 | 0.50 |

All of the models were shown to satisfy the properties checked against them and to be free of deadlock. Statistics concerning the verification results are shown in Table I. These analyses indicated that, for all of the evaluated applications, the generation method was fulfilling its intended purpose: generating interface designs that will always satisfy the human operator task models.

## VIII. Discussion and Future Work

The work presented in this paper described a novel approach for using L* learning to automatically generate functional human–machine interface designs. 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 by a task analysis and thus supports UCD. This finding is supported by the presented validation results that show that the generated interface designs allow every part of a task to execute and do so without ever preventing tasks from finishing or locking the user out of the system. Furthermore, because L* learning ensures that a minimal model is produced, the state space of the interface will also be minimal [11], [46].

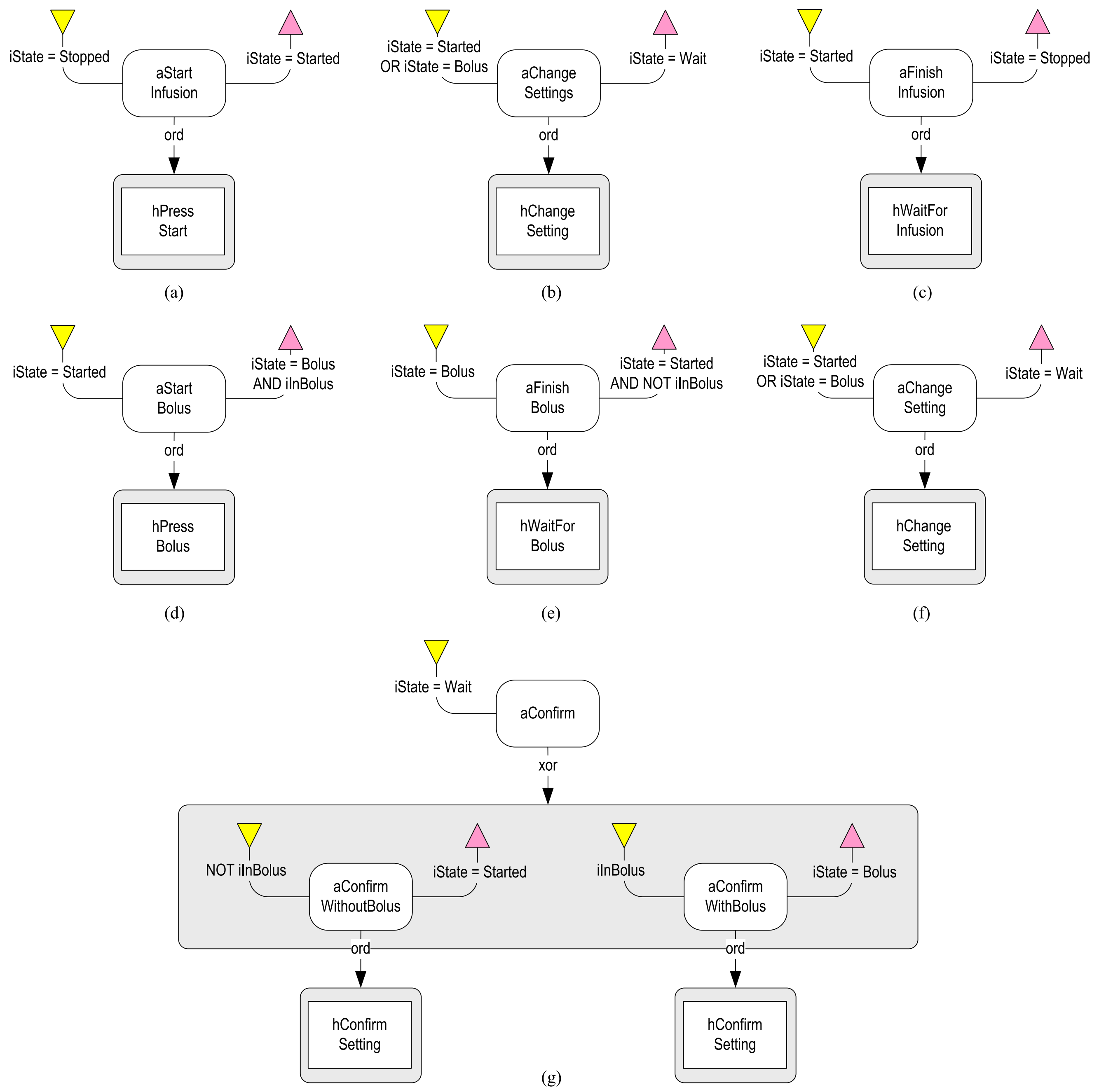


Fig. 6. Task models describing how the human should behave when interacting with the PCA pump.

This is a potentially advantageous property because it ensures the minimal complexity of the interface [56].

While all of the presented applications demonstrate the power of our method, the PCA pump is particularly compelling. This is because the generation method was able to create the same interface that was designed for the PCA pump reference model [64] based on human task behavior the device was intended to support. This is important because it shows that our method is capable of generating a formal interface design that complies with the interface the reference standard was rigorously verified with. This suggests that our method is appropriate for safety critical applications.

Despite these advances, there are still limitations of the methods and ways that it could be developed and assessed in the future. These are explored in the discussion below.

## A. *Interface Usability and Other Artifacts of UCD*

An interesting feature of our method is that it creates interfaces with forcing functions. That is, it creates interfaces that will not let the human operator go off task [65]. For example, the vending machine application will not let the human operator enter more than two quarters into the machine. This is a known method for ensuring safe system operation and preventing

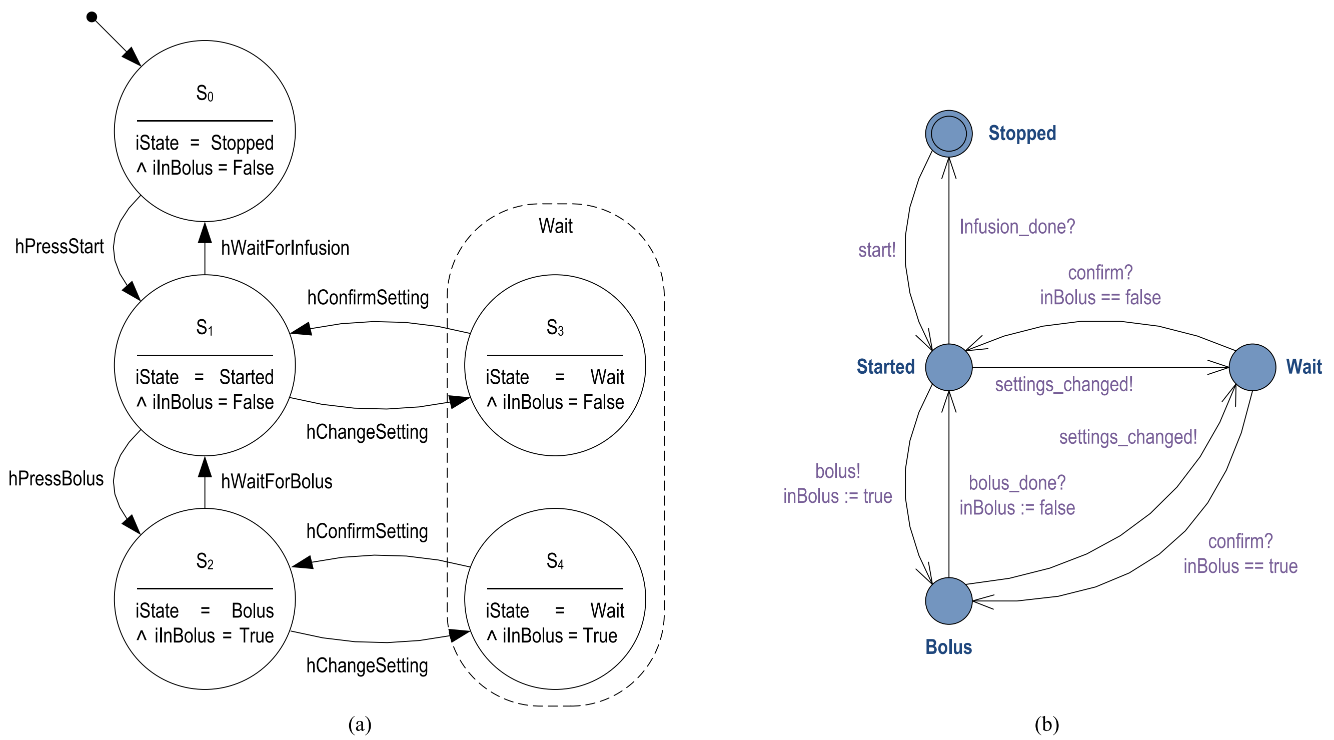


Fig. 7. (a) PCA pump interface design generated from the task model shown in Fig. 6. (b) PCA pump interface reference model from [64], presented in its original notation. Note that in (b), states have been slightly rearranged to support easy comparison with the interface in (a). States and actions have been slightly renamed for the same purpose. Recursive transitions to states (transitions that point to the same state they originated from) have been removed because they relate to automation behavior that was not of concern in this analysis. Further note that in (b), multiple expressions on a transition are assumed to be in an "and" relationship. == represents a Boolean equals and := represents a variable assignment.

human error. However, it also results in a system that could be rigid or inflexible in unusual or unanticipated situations. Future work should investigate when the forcing function feature of our method is inappropriate and explore methods for accounting for human operator tasks differently.

It is important to note that even though the generation method ensures the use of forcing functions, this does not mean that the generation method forbids nondeterminism in human tasks. To the contrary, nondeterminism is inherently built into the different decomposition operators of EOFM: allowing human tasks to be described with multiple paths to success [20]. Such nondeterminism is present in the described applications. For example, in the drink machine, the human can perform any of the three tasks in Fig. 4 at any time, as long as the precondition of the top-most activity of each task is satisfied. Thus, while there are some potential drawbacks to the use of forcing functions, they do not inherently remove the flexibility often associated with valid task models.

In the extended formal human–automation interaction literature [4], there are a number of different formal usability properties [66]–[69]. These relate to the reachability of interface states, the visibility of interface states, task-related properties (properties concerning task compatibility), and reliability [4]. We know from the generation process and the verification analyses that our method supports task-related usability properties. However, it is not clear if the generated interface will satisfy the other types of formal processes.

Our method 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.

Work by Bowen and Reeves [6] applied formal methods to other artifacts commonly generated during UCD, such as interface drawings and storyboard concepts. Specifically, they developed techniques that allowed formal methods to describe the behavior interface prototypes so that they can be used in formal refinement processes (discussed in Section VIII-C) and other types of formal analyses.

Future work should investigate whether these other formal usability and UCD considerations can be accounted for in our interface generation process to make it more robust and useful to designers.

## B. Scalability

We have made many efforts to ensure that the generation process is fast, efficient, and will scale. This includes using the state-space efficient version of the EOFM to SAL translator in the teacher oracle [38]. We also reduce model-checking time by using the database to allow the oracle to only check the part of the query not previously processed.

These efforts are reflected in the results. As the number of tasks grew from two in the light switch application, to three in the vending machine, and to seven in the PCA pump, the amount of time required to learn the interface went up. However, even the application that took the longest to generate the PCA pump interface took less than a minute and a half. Thus, the scalability was never a limitation for the examined applications. However, because the method uses model checking, it is subject to the state explosion problem: a situation where model size can quickly grow too big to check [3]. Scalability has shown itself to be a major limiting factor when using model checking to evaluate human-interactive systems that require data entry [25]. Thus, it is likely that scalability will be a limitation to the method for more complex applications. Future work should explore additional ways of improving the method's scalability.

## C. Formal Refinement

Refinement is a process in which a formal description of a system is transformed into another one while ensuring that properties are maintained [70]. Refinement has shown promise for converting formal human–computer interface descriptions into actual interface implementations [71]. This concept is similar to the interface generation discussed previously in Section II-F. However, where the previous work was concerned with shortening development time, refinement strives to provide formal guarantees about performance and usability.

Because EOFM task models have formal semantics [20], it may be possible to generate formal interface designs directly from task models by applying formal refinement to these semantics. Because refinement processes do not rely on model checking, they may be more computationally efficient than the L* learning approach we employed. It is important to note, however, that EOFM can have a significant amount of nondeterminism in the order that activities and actions can be executed in. This flexibility is advantageous for modelers because it ensures that they are not overconstraining the normative behavior of the human operator or the assumptions about the system they are interacting with. However, creating transformation rules for converting EOFM tasks into interface designs through formal refinements that account for this nondeterminism may not be straightforward.

Santoro [52] (whose work was briefly discussed under Section II-F) developed techniques for generating abstract descriptions of interfaces from formal task models represented in CTT using transformation rules and design heuristics. However, these abstractions required additional designer effort to ultimately produce finalized designs. The CTT approach can be used to provide formal guarantees, but these are done by checking the CTT model itself or by integrating it with other elements of a larger formal system model and then performing formal verification. As such, research has not yet identified the transformation rules that will preserve the task performance requirements encapsulated by the task models in a generated interface (though the heuristics used in [47]–[52] would be good initial candidates) or the concurrent usability properties we plan to incorporate in future developments (see Section VIII-A). Because nobody has done this before, it is not clear how difficult such a process will be. It could be that the automated learning afforded by our use of the L* algorithm will prove to facilitate a much more straightforward process. Future work should explore the possibility of using formal refinement techniques with EOFM to create interface designs. This refinement process could then be compared to the learning approach presented here based on their scalability, usability, and the guarantees they can provide.

## D. Task Modeling for Interface Generation

In creating the task-analytic models for the presented application, two 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 ensured that the 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 in the vending machine application, the task model was required to note the amount of money entered into the machine before inserting an additional quarter. The task's completion condition also had to assert that the amount of money registered in the machine was increased by 25¢. These constraints can certainly work for some applications, like the ones presented here. However, they may be incompatible for systems with complex tasks. Future work should investigate how to avoid these constraints.

## E. Validation With Other Applications and Human Subjects

The applications presented in this paper are illustrative, but simple. We plan to continue developing applications as the generation method evolves to test the applicability of our method in different domains.

It is important to note that applications presented here were meant to test and demonstrate the capabilities of the method. They do not necessarily represent the best practices for task analysis. Clearly, the success of interfaces generated with the method depends on the quality of the task models used as input. Ideally, task models would be derived from a cognitive task analysis conducted before the engineering of a system. This would allow the generation method to be genuinely used as part of UCD. However, because the presented applications are simple and based on the existing analyses [64], the authors feel confident that the task models used in the presented generations are valid and reasonably representative.

While we have confidence in the ability of the method to generate successful descriptions of interface behavior from task models, the method could be more thoroughly validated. In particular, future work should focus on using the method to generate interfaces for systems that are actually being designed using task models and UCD. Such an effort would start with a task analysis through which the task behavior model and other display considerations would be discovered. The generation method would then be used to create the functional description of the human–machine interface. This would be used in conjunction with the other requirements elucidated in the cognitive task analysis and usability design principles to create an interface. Human subject experiments would then be used to assess how usable the generated interface is. Such efforts will be the subject of future research.

## References

[1] M. Li, K. Molinaro, and M. L. Bolton, "Learning formal human–machine interface designs from task analytic models," in *Proc. Human Factors Ergonom. Soc. Annu. Meeting*, 2015, pp. 652–656.

[2] *Ergonomics of Human System Interaction—Part 210: Human-Centred Design for Interactive Systems*, ISO 9241–210: 2010.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[4] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using formal verification to evaluate human-automation interaction: A review," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 3, pp. 488–503, May 2013.

[5] J. C. Campos and M. Harrison, "Formally verifying interactive systems: A review," in *Proc. 4th Int. Eurograph. Workshop Des., Specification, Verification Interact. Syst.*, 1997, pp. 109–124.

[6] J. Bowen and S. Reeves, "Formal models for user interface design artefacts," *Innovations Syst. Softw. Eng.*, vol. 4, no. 2, pp. 125–141, 2008.

[7] C. Heitmeyer, "On the need for practical formal methods," in *Proc. 5th Int. Symp. Formal Techn. Real-Time Fault-Tolerant Systems*, 1998, pp. 18–26.

[8] M. L. Bolton and S. Ebrahimi, "An approach to generating human-computer interfaces from task models," in *Proc. AAAI Spring Symp. Ser.*, 2014, pp. 92–97.

[9] J. M. Schraagen, S. F. Chipman, and V. L. Shalin, *Cognitive Task Analysis*. Philadelphia, PA, USA: Lawrence Erlbaum Assoc., Inc., 2000.

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

[11] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

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

[13] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Syst. Tech. J.*, vol. 34, no. 5, pp. 1045–1079, 1955.

[14] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Stud.*, vol. 34, pp. 129–153, 1956.

[15] F. Paternò, "Model-based design of interactive applications," *Intelligence*, vol. 11, no. 4, pp. 26–38, 2000.

[16] R. W. Chu, C. M. Mitchell, and P. M. Jones, "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. A, Syst. Humans*, vol. 25, no. 7, pp. 1054–1075, Jul. 1995.

[17] A. Lecerof and F. Paternò, "Automatic support for usability evaluation," *IEEE Trans. Softw. Eng.*, vol. 24, no. 10, pp. 863–888, Oct. 1998.

[18] E. J. Bass, S. T. Ernst-Fortin, R. L. Small, and J. Hogans, "Architecture and development environment of a knowledge-based monitor that facilitate incremental knowledge-base development," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 34, no. 4, pp. 441–449, Jul. 2004.

[19] M. L. Bolton and E. J. Bass, "A method for the formal verification of human interactive systems," in *Proc. 53rd Annu. Meeting Human Factors Ergonom. Soc.*, 2009, pp. 764–768.

[20] M. L. Bolton, R. I. Siminiceanu, and E. J. Bass, "A systematic approach to model checking human-automation interaction using task-analytic models," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 41, no. 5, pp. 961–976, Sep. 2011.

[21] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Generating phenotypical erroneous human behavior to evaluate human–automation interaction using model checking," *Int. J. Human-Comput. Stud.*, vol. 70, pp. 888–906, 2012.

[22] M. L. Bolton and E. J. Bass, "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.: Syst., vol. 43, no. 6, pp. 1314–1327, Nov. 2013.

[23] K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey, "A survey of user-centered design practice," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.: Changing Our World, Changing Ourselves.*, 2002, pp. 471–478.

[24] Y. Aït-Ameur, M. Baron, and P. Girard, "Formal validation of HCI user tasks," in *Proc. Int. Conf. Softw. Eng. Res. Practice*, 2003, pp. 732–738.

[25] M. L. Bolton and E. J. Bass, "Formally verifying human-automation interaction as part of a system model: Limitations and tradeoffs," *Innov. Syst. Softw. Eng.*, vol. 6, no. 3, pp. 219–231, 2010.

[26] F. Paternò and C. Santoro, "Integrating model checking and HCI tools to help designers verify user interface properties," in *Proc. 7th Int. Workshop Des., Specification, Verification Interact. Syst.*, 2001, pp. 135–150.

[27] P. Palanque and F. Paternò, Eds., *Formal Methods in Human-Computer Interaction*. Secaucus, NJ, USA: Springer, 1997.

[28] D. Navarre, P. Palanque, F. Paternò, C. Santoro, and R. Bastide, "A tool suite for integrating task and system models through scenarios," in *Proc. Int. Workshop Des., Specification, Verification Interact. Syst.*, 2001, pp. 88–113.

[29] E. Barboni, J. -F. Ladry, D. Navarre, P. Palanque, and M. Winckler, "Beyond modelling: An integrated environment supporting co-execution of tasks and systems models," in *Proc. 2nd ACM SIGCHI Symp. Eng. Interact. Comput. Syst.*, 2010, pp. 165–174.

[30] S. Basnyat, P. Palanque, B. Schupp, and P. Wright, "Formal socio-technical barrier modelling for safety-critical interactive systems design," *Safety Sci.*, vol. 45, no. 5, pp. 545–565, 2007.

[31] E. L. Gunter, A. Yasmeen, C. A. Gunter, and A. Nguyen, "Specifying and analyzing workflows for automated identification and data capture," in *Proc. 42nd Hawaii Int. Conf. Syst. Sci.*, 2009, pp. 1–11.

[32] J. C. Campos, "Using task knowledge to guide interactor specifications analysis," in *Proc. 10th Int. Workshop Interact. Syst.. Des., Specification, Verification*, 2003, pp. 171–186.

[33] M. L. Bolton, N. Jimenez, M. M. van Paassen, and M. Trujillo, "Formally verifying human-automation interaction with specification properties generated from task analytic models," in *Proc. 6th IAASS Conf.*, 2013, [CD-ROM], 8 pp.

[34] M. L. Bolton, "Automatic validation and failure diagnosis of human-device interfaces using task analytic models and model checking," *Comput. Math. Org. Theory*, vol. 19, no. 3, pp. 288–312, 2013.

[35] F. Paternò, C. Mancini, and S. Meniconi, "Concurtasktrees: A diagrammatic notation for specifying task models," in *Proc. IFIP TC13 Int. Conf. Human-Comput. Interact.*, 1997, pp. 362–369.

[36] C. M. De Almeida, P. Palanque, M. Ragosta, and R. M. Fahssi, "Extending procedural task models by explicit and systematic integration of objects, knowledge and information," in *Proc. 31st Eur. Conf. Cogn. Ergonom.*, 2013, Art. no. 23.

[37] M. Giese, T. Mistrzyk, A. Pfau, G. Szwillus, and M. von Detten, "AMBOSS: A task modeling approach for safety-critical systems," in *Proc. 2nd Int. Conf. Human-Centered Softw. Eng.*, 2008, pp. 98–109.

[38] M. L. Bolton, X. Zheng, K. Molinaro, A. Houser, and M. Li, "Improving the scalability of formal human–automation interaction verification analyses that use task-analytic models," in *Innovations in Systems and Software Engineering*, vol. 13. New York, NY, USA: Springer, 2016.

[39] M. L. Bolton and E. J. Bass, "Using relative position and temporal judgments to identify biases in spatial awareness for synthetic vision systems," *Int. J. Aviat. Psychol.*, vol. 18, no. 2, pp. 1050–8414, 2008.

[40] M. L. Bolton, N. Jimenez, M. M. van Paassen, and M. Trujillo, "Automatically generating specification properties from task models for the formal verification of human-automation interaction," *IEEE Trans. Human–Mach. Syst.*, vol. 44, no. 5, pp. 561–575, Oct. 2014.

[41] M. L. Bolton, "Model checking human–human communication protocols using task models and miscommunication generation," *J. Aerosp. Inf. Syst.*, vol. 12, no. 7, pp. 476–489, 2015.

[42] M. L. Bolton and E. J. Bass, "Using model checking to explore checklist-guided pilot behavior," *Int. J. Aviat. Psychol.*, vol. 22, pp. 343–366, 2012.

[43] D. Pan and M. L. Bolton, "Properties for formally assessing the performance level of human-human collaborative procedures with miscommunications and erroneous human behavior," *Int. J. Ind. Ergonom.*, to be published, [Online]. Available: https://doi.org/10.1016/j.ergon.2016.04.001

[44] A. Abbate, A. Throckmorton, and E. Bass, "A formal task-analytic approach to medical device alarm troubleshooting instructions," *IEEE Trans. Hum.-Mach. Syst.*, vol. 46, no. 1, pp. 53–65, Feb. 2016.

[45] M. L. Bolton and E. J. Bass, "Using task analytic models to visualize model checker counterexamples," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2010, pp. 2069–2074.

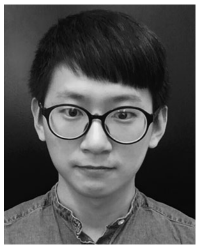
[46] H. Raffelt, B. Steffen, and T. Berg, "Learnlib: A library for automata learning and experimentation," in *Proc. 10th Int. Workshop Formal Methods Ind. Critical Syst.*, 2005, pp. 62–71.

[47] V. Tran, M. Kolp, J. Vanderdonckt, Y. Wautelet, and S. Faulkner, "Agent-based user interface generation from combined task, context and domain models," in *Proc. 8th Int. Workshop Task Models Diagrams User Interface Des.*, 2010, pp. 146–161.

[48] J. G. García, C. Lemaigre, J. M. González-Calleros, and J. Vanderdonckt, "Model-driven approach to design user interfaces for workflow information systems." *J. Universal Comput. Sci.*, vol. 14, no. 19, pp. 3160–3173, 2008.

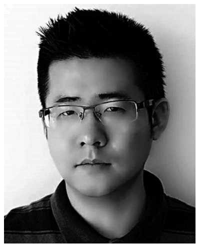
[49] A. Mahfoudhi, M. Abid, and M. Abed, "Towards a user interface generation approach based on object oriented design and task model," in *Proc. 4th Int. Workshop Task Models Diagrams*, 2005, pp. 135–142.

[50] S. España, I. Pederiva, and J. I. Panach, "Integrating model-based and task-based approaches to user interface generation," in *Computer-Aided Design of User Interfaces V*. Dordrecht, The Netherlands: Springer, 2007, pp. 253–260.
[51] K. Luyten, T. Clerckx, K. Coninx, and J. Vanderdonckt, "Derivation of a dialog model from a task model by activity chain extraction," in *Proc. 10th Int. Workshop Interact. Syst. Design, Specification, Verification*. Berlin, Germany: Springer, 2003, pp. 203–217.
[52] C. Santoro, *A Task Model-Based Approach for Design and Evaluation of Innovative User Interfaces*. Louvain-la-Neuve, Belgium: Presses Univ. Louvain, 2005.
[53] M. Heymann and A. Degani, "Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm," *Human Factors*, vol. 49, no. 2, pp. 311–330, 2007.
[54] N. B. Sarter and D. D. Woods, "How in the world did we ever get into that mode? Mode error and awareness in supervisory control," *Human Factors*, vol. 37, no. 1, pp. 5–19, 1995.
[55] A. Degani and A. Kirlik, "Modes in human-automation interaction: Initial observations about a modeling approach," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, 1995, vol. 4, pp. 3443–3450.
[56] S. Combéfis, D. Giannakopoulou, C. Pecheur, and M. Feary, "Learning system abstractions for human operators," in *Proc. Int. Workshop Mach. Learn. Technol. Softw. Eng.*, 2011, pp. 3–10.
[57] S. Combéfis and C. Pecheur, "Automatic generation of full-control system abstraction for human–machine interaction," in *Proc. Human–Mach. Interact. (Formal H)*, 2012, p. 9.
[58] S. Combéfis and C. Pecheur, "A bisimulation-based approach to the analysis of human-computer interaction," in *Proc. 1st ACM SIGCHI Symp. Eng. Int. Comput. Syst.*, 2009, pp. 101–110.
[59] A. Degani, *Taming HAL: Designing Interfaces Beyond 2001*. New York, NY, USA: Macmillan, 2004.
[60] E. Palmer, ""Oops, it didn't arm"—A case study of two automation surprises," in *Proc. 8th Int. Symp. Aviation Psychol.*, 1995, pp. 227–232.
[61] J. Rushby, "The versatile synchronous observer," in *Formal Methods: Foundations and Applications* (ser. Lecture Notes in Computer Science), vol. 7498, R. Gheyi and D. Naumann, Eds. Berlin, Germany: Springer, 2012.
[62] J. Carroll and D. Long, *Theory of Finite Automata with an Introduction to Formal Languages*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.
[63] L. De Moura, S. Owre, and N. Shankar, "The SAL language manual," Comput. Sci. Lab., , SRI Int., Menlo Park, CA, USA, Tech. Rep. CSL-01–01, 2003.
[64] D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky, "Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project," in *Proc. Joint Workshop High Confidence Med. Devices, Softw., Syst. Med. Device Plug-and-Play Interoperability*, 2007, pp. 23–33.
[65] D. A. Norman, *The Psychology of Everyday Things*. New York, NY, USA: Basic Books, 1988.
[66] J. C. Campos and M. D. Harrison, "Interaction engineering using the ivy tool," in *Proc. 1st ACM SIGCHI Symp. Eng. Interact. Comput. Syst.*, 2009, pp. 35–44.
[67] J. C. Campos and M. D. Harrison, "Systematic analysis of control panel interfaces using formal tools," in *Proc. 15th Int. Workshop Design, Verification Specification Interactive Syst.*, 2008, pp. 72–85.
[68] G. D. Abowd, H. Wang, and A. F. Monk, "A formal technique for automated dialogue development," in *Proc. 1st Conf. Designing Interact. Syst.*, 1995, pp. 219–226.
[69] F. Paternò, "Formal reasoning about dialogue properties with automatic support," *Interact. Comput.*, vol. 9, no. 2, pp. 173–196, 1997.
[70] N. Wirth, "Program development by stepwise refinement," *Commun. ACM*, vol. 14, no. 4, pp. 221–227, 1971.
[71] J. Bowen and S. Reeves, "Refinement for user interface designs," *Formal Aspects Comput.*, vol. 21, no. 6, pp. 589–612, 2009.



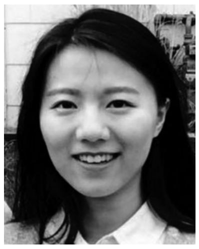
**Meng Li** received the M.S. degree in industrial engineering in 2016 from the University at Buffalo, The State University of New York, Amherst, NY, USA, where he is currently working toward the Ph.D. degree in industrial engineering.

His research interests include human–computer interaction, human behavior modeling, information displays, and formal verification.



**Jiajun Wei** received the M.S. degree in applied psychology (focusing on engineering psychology) from Zhejiang University, Hangzhou, China, in 2014. He is currently working toward the Ph.D. degree in industrial and systems engineering at the University at Buffalo, The State University of New York, Amherst, NY, USA.

His research interests include human factor engineering, human–computer interaction, cognition, judgment, and decision making.



**Xi Zheng** received the B.S. degree in electronic commerce from Southwest University, Chongqing, China, in 2011. She is currently working toward the Ph.D. degree in the Department of Industrial and Systems Engineering, University at Buffalo, The State University of New York, Amherst, NY, USA.

Her research focuses on the use of formal methods and human reliability analyses to understand and reduce medical errors.

**Matthew L. Bolton** (S'05–M'10–SM'16) received the B.S. degree in computer science, the M.S. degree in systems engineering, and the Ph.D. degree in systems engineering from the University of Virginia, Charlottesville, VA, USA, in 2004, 2006, and 2010, respectively.

He is an Assistant Professor with the Department of Industrial and Systems Engineering, University at Buffalo, The State University of New York, Amherst, NY, USA. His research focuses on the use of human performance modeling and formal methods in the analysis, design, and evaluation of safety-critical systems.