Published in final edited form as:
Innov Syst Softw Eng. 2010 March 25; 6(3): 219–231. doi:10.1007/s11334-010-0129-9.

Formally verifying human–automation interaction as part of a system model: limitations and tradeoffs

Matthew L. Bolton and **Ellen J. Bass**
Department of Systems and Information Engineering, University of Virginia, 151 Engineer's Way, Charlottesville, VA, USA

Matthew L. Bolton: mlb4b@virginia.edu; Ellen J. Bass: ejb4n@virginia.edu

Abstract

Both the human factors engineering (HFE) and formal methods communities are concerned with improving the design of safety-critical systems. This work discusses a modeling effort that leveraged methods from both fields to perform formal verification of human–automation interaction with a programmable device. This effort utilizes a system architecture composed of independent models of the human mission, human task behavior, human-device interface, device automation, and operational environment. The goals of this architecture were to allow HFE practitioners to perform formal verifications of realistic systems that depend on human–automation interaction in a reasonable amount of time using representative models, intuitive modeling constructs, and decoupled models of system components that could be easily changed to support multiple analyses. This framework was instantiated using a patient controlled analgesia pump in a two phased process where models in each phase were verified using a common set of specifications. The first phase focused on the mission, human-device interface, and device automation; and included a simple, unconstrained human task behavior model. The second phase replaced the unconstrained task model with one representing normative pump programming behavior. Because models produced in the first phase were too large for the model checker to verify, a number of model revisions were undertaken that affected the goals of the effort. While the use of human task behavior models in the second phase helped mitigate model complexity, verification time increased. Additional modeling tools and technological developments are necessary for model checking to become a more usable technique for HFE.

Keywords

Human–automation interaction; Task analysis; Formal methods; Model checking; Safety critical systems; PCA pump

1 Introduction

Both human factors engineering (HFE) and formal methods are concerned with the engineering of robust systems that will not fail under realistic operating conditions. The traditional use of formal methods has been to evaluate a system's automation under different operating and/or environmental conditions. However, human operators control a number of safety critical systems and contribute to unforeseen problems. For example, human behavior

Correspondence to: Matthew L. Bolton, `mlb4b@virginia.edu`.

has contributed to between 44,000 and 98,000 deaths nationwide every year in medical practice [18], 74% of all general aviation accidents [19], at least two-thirds of commercial aviation accidents [28], and a number of high profile disasters such as the incidents at Three Mile Island and Chernobyl [22]. HFE focuses on understanding human behavior and applying this knowledge to the design of human–automation interaction: making systems easier to use while reducing errors and/or allowing recovery from them [25,29].

By leveraging the knowledge of both HFE and formal methods, researchers have identified the cognitive preconditions for mode confusion and automation surprise [7,10, 16,23]; automatically generated user interface specifications, emergency procedures, and recovery sequences [13,14]; and identified human behavior sequences (normative or erroneous) that contribute to system failures [8,12].

While all of this work has produced useful results, the models have not included all of the components necessary to analyze human–automation interaction. For HFE analyses of human–automation interaction, the minimal set of components are the goals and procedures of the human operator; the automated system and its human interface; and the constraints imposed by the operational environment. Cognitive work analysis is concerned with identifying constraints in the operational environment that shape the mission goals of the human operator [27]; cognitive task analysis is concerned with describing how human operators normatively and descriptively perform goal oriented tasks when interacting with an automated system [17,24]; and modeling frameworks such as [11] seek to find discrepancies between human mental models, human-device interfaces (HDIs), and device automation. In this context, problems related to human–automation interaction may be influenced by the human operator's mission, the human operator's task behavior, the operational environment, the HDI, the device's automation, and their interrelationships.

We are developing methods and tools to allow human factors engineers to exploit their existing human task modeling constructs with the powerful verification capabilities of formal methods in order to identify potential problems with human–automation interaction in safety critical systems that may be related to human task behavior, the automated device, the operational environment, or their interaction. To this end, we are developing a computational framework (Fig. 1) for the formal modeling of human-automation interaction. This framework utilizes concurrent models of human operator task behavior, human mission (the goals the operator wishes to achieve using the system), device automation, and the operational environment which are composed together to form a larger system model. Inter-model interaction is represented by variables shared between models. Environment variables communicate information about the state of the environment to the device automation, mission, and human task models. Mission variables communicate the mission goals to the human task model. Interface variables convey information about the state of the HDI (displayed information, the state of input widgets, etc.) to the human task model. The human task model indicates when and what actions a human operator would perform on the HDI. The HDI communicates its current state to the device automation via the interface variables. The HDI receives information about the state of the device automation model via the automation state variables.

For broader applicability, the analysis framework must support modeling constructs intuitive to the human factors engineer in order to allow him to effectively model human missions, human tasks, and HDIs. Because an engineer may wish to rerun verifications using different missions, task models, HDIs, environments, or automation behaviors, these components should remain decoupled (as is the case in Fig. 1). Finally, the modeling technique must be capable of representing the target systems with enough fidelity to allow the engineer to perform the desired verification, and do so in a reasonable amount of time (this could mean several hours for a small project, or several days for a more complicated one).

This paper describes an instantiation of this framework using a model of a Baxter Ipump Pain Management System [2], a patient controlled analgesia (PCA) pump that administers pain medication in accordance with constraints defined by a health care technician (described in Sect. 2.1). Models were developed in two phases. The first phase involved the construction and debugging of the HDI, device automation, and human mission models (an environmental model was not included because of the general stability of the environment in which an actual pump operates) with an unconstrained human task model serving as a placeholder for a more realistic human task model. The second extended the model produced in Phase 1 with a realistic model of the human task, completing the framework.

Even though the target device in this modeling effort was seemingly simple, the system model that was initially developed in Phase 1 (Phase 1a) was too difficult for the model checker to process quickly and too complex for it to verify. Thus a number of revisions were undertaken [3]. In Phase 1b a reduced and abstracted model of the Baxter Ipump was produced which, while capable of being used in some verifications, did so at the expense of limiting the number of goals represented in the human mission model. This Phase 1b model limited the usefulness of incorporating human task behavior in Phase 2. Thus, in Phase 1c, the system model was reduced to encompass the programming procedure for a much simpler PCA pump. In Phase 2, the incorporation of the more realistic human task behavior actually resulted in a reduction of the total system model’s complexity, but did so at the expense of an increase in verification time. This paper discusses these modeling phases, the verification results produced in them, and their associated compromises in relation to the goals of the modeling architecture. These are used to draw conclusions about the feasibility of using formal methods to inform human–automation interaction.

2 Methods

2.1 The target system

The Baxter Ipump is an automated machine that controls delivery of sedative, analgesic, and anesthetic medication solutions [2]. Solution delivery via intravenous, subcutaneous, and epidural routes is supported. Medication solutions are typically stored in bags locked in a compartment on the back of the pump.

Pump behavior is dictated by internal automation, which can depend on how the pump is programmed by a human operator. Pump programming is accomplished via its HDI (Fig. 2) which contains a dynamic LCD display, a security key lock, and eight buttons. When programming the pump, the operator is able to specify all of the following: whether to use periodic or continuous doses of medications (i.e., the mode which can be PCA, Basal+PCA, or Continuous), whether to use prescription information previously programmed into the pump, the fluid volume contained in the medication bag, the units of measure used for dosage (ml, mg, or μg), whether or not to administer a bolus (an initial dose of medication), dosage amounts, dosage flow rates (for either basal or continuous rates as determined by the mode), the delay time between dosages, and 1 h limits on the amount of delivered medication.

During programming, the security key is used to lock and unlock the compartment containing the medication solution. The unlocking and locking process is also used as a security measure to ensure that an authorized person is programming the pump. The start and stop buttons are used to start and stop the delivery of medication at specific times during programming. The on–off button is used to turn the device on and off.

The LCD display supports pump operation options. When the operator chooses between two or more options, the interface message indicates what is being chosen, and the initial or default option is displayed. Pressing the up button allows the programmer to scroll through the available options.

When a numerical value is required, the interface message conveys its name and the displayed value is presented with the cursor under one of the value's digits. The programmer can move the position of the cursor by pressing the left and right buttons. He or she can press the up button to scroll through the different digit values available at that cursor position. The clear button sets the displayed value to zero. The enter button is used to confirm values and treatment options.

Aside from the administration of treatment, the pump's automation supports dynamic checking and restriction of operator entered values. Thus, in addition to having hard limits on value ranges, the extrema can change dynamically in response to other user specified values.

2.2 Apparatus

All formal models were constructed using the Symbolic Analysis Laboratory (SAL) language [9] because of its associated analysis and debugging tools, and its support for both the asynchronous and synchronous composition of different models (modules using SAL's internal semantics). The task model representations described next were translated into the SAL language as a single module using a custom-built java program [5]. All verifications were done using SAL-SMC 3.0, the SAL symbolic model checker.[1] Verifications were conducted on a 3.0 GHz dual-core Intel Xeon processor with 16 GB of RAM running the Ubuntu 9.04 desktop.

Human task models were created using an intermediary language called enhanced operator function model (EOFM) [4], an XML-based, generic human task modeling language based on the operator function model (OFM) [21,26]. EOFMs are hierarchical and heterarchical representations of goal or mission driven activities that decompose into lower level activities, and finally, atomic actions—where actions can represent any observable, cognitive, or perceptual human behavior. EOFMs express task knowledge by explicitly specifying the conditions under which human operator activities can be undertaken: what must be true before they can execute (preconditions), when they can repeat (repeat conditions), and when they have completed (completion conditions). Any activity can decompose into one or more other activities or one or more actions. A decomposition operator specifies the temporal relationships between and the cardinality of the decomposed activities or actions (when they can execute relative to each other and how many can execute).

EOFMs can be represented visually as a tree-like graph structure (examples can be seen in Figs. 3, 4, 5, 6, 7). In these representations, actions are represented as rectangles and activities are represented as rounded rectangles. An activity's decomposition is presented as an arrow, labeled with the decomposition operator, extending below it that points to a large rounded rectangle containing the decomposed activities or actions. In this work, three decomposition operators are used: (1) ord (all activities or actions in the decomposition must execute in the order they appear); (2) or_seq (one or more of the activities or actions in the decomposition must execute); and (3) xor (exactly one activity or action in the decomposition must execute). Conditions on activities are represented as shapes or arrows

[1]Some model debugging was also conducted using SAL's bounded model checker.

(annotated with the condition logic) connected to the activity that they constrain. The form, position, and color of the shape are determined by the type of condition. A precondition is presented as a yellow, downward-pointing triangle connected to the left side of the activity. A completion condition is represented by a magenta, upward-pointing triangle connected to the right side of the activity. A repeat condition is depicted as an arrow recursively pointing to the top of the activity. More details can be found in [4].

2.3 Verification specification

Two specifications were employed in each of the modeling phases: both were written in linear temporal logic and evaluated using SAL–SMC. The first (Eq. 1), used for model debugging, verifies that a valid prescription could be programmed into the pump:

$$\mathbf{G}\neg\left(\begin{array}{lccc} & iInterfacemessage & = & \text{TreatmentAdministering} \\ \wedge & iMode & = & iPrescribedMode \\ \wedge & lFluidVolume & = & iPrescribedFluidVolume \\ \wedge & lPCADose & = & iPrescribedPCADose \\ \wedge & lDelay & = & iPrescribedDelay \\ \wedge & lBasalRate & = & iPrescribedBasalRate \\ \wedge & l1HourLimit & = & iPrescribed1HourLimit \\ \wedge & lBolus & = & iPrescribedBolus \\ \wedge & lContinuousRate & = & iPrescribedContinuousRate \end{array}\right) \quad (1)$$

Here, if the model is able to enter a state indicating that treatment is administering (*iInterfaceMessage* = *TreatmentAdministering*) with the entered (or programmed) prescription values (*iMode*, *lFluidVolume*,…, *lContinuous-Rate*) matching the prescription values generated by the mission model (variables with the *iPrescribed* prefix), a counterexample would be produced illustrating how that prescription was programmed. Variables with an *i* prefix indicate that the variable is an input to the human task model. Variables with an *l* prefix indicate that the variable is local to a given model.

The second specification (Eq. 2) represented a safety property that was expected to verify to true, thus allowing the model checker to traverse the entire state space of each phase's model. Because such a verification allows SAL to report the size of a model's state space, verifications using this specification would provide some means of comparing the complexity of the models produced in each phase.

$$\mathbf{G}\neg\left(\begin{array}{lccc} & iInterfaceMessage & = & \text{TreatmentAdministering} \\ \wedge & iMode & \neq & \text{Continuous} \\ \wedge & lDelay & = & 0 \end{array}\right) \quad (2)$$

Here, the specification is asserting that the model should never enter a state where treatment is administering in the PCA or Basal+PCA modes (*iMode* ≠ Continuous) when there is no delay between doses.[2] Thus, if Eq. (2) verifies to true, the pump will never allow a programmer to enter prescriptions that would allow patients to continuously administer PCA doses to themselves [2].

[2]A delay can only been set when the PCA or Basal + PCA modes have been selected by the human operator. There are no delays between doses when the pump is in the Continuous mode.

3 Phase 1a: a representative model of the Ipump

3.1 Model description

An initial model was created to conform to the architectural and design philosophy represented in Fig. 2: the mission was represented as a set of viable prescriptions options; the mission, human operator, human-device interface, and device automation were modeled independently of each other; and the behavior of the automated system and HDI models was designed to accurately reflect the behavior of these systems as described in the user's manual [2] and observed through direct interaction with the device. An unconstrained human operator was constructed that could issue any valid human action to the human-device interface model at any given time. Because the PCA pump generally operates in a controlled environment, away from temperature and humidity conditions that might affect the performance of the pump's automation, no environmental model was included. Finally, because documentation related to the internal workings of the pump was limited, the system automation model was restricted to that associated with the pump programming procedure: behavior that could be gleaned from the operator's manual [2], correspondences with hospital staff, and direct interaction with the pump.

3.2 Model coordination

Model infrastructure was required to ensure that human operator actions were properly recognized by the HDI model. In an ideal modeling environment, human action behavior originating from the human operator model could have both an asynchronous and synchronous relationship with the HDI model. Synchronous behavior would allow the HDI model to react to user actions in the same transition in which they were issued/performed by the human operator model. However, both the human operator and HDI models operate independently of each other, and may have state transitions that are dependent on internal or external conditions that are not directly related to the state of the other model. This suggests an asynchronous relationship. SAL only allows models to be composed with each other asynchronously or synchronously (but not both). Thus, it was necessary to adapt the models to support features associated with the unused composition.

Asynchronous composition was used to compose the human operator and HDI models. This necessitated some additional infrastructure to prevent the human operator model from issuing user inputs before the HDI model was ready to interpret them and to prevent the human operator model from terminating a given input before the interface could respond to it. This was accomplished through the addition of two Boolean variables: one indicating that input had been submitted (henceforth called *Submitted*) and a variable indicating the interface was ready to receive actions (henceforth called *Ready*). This coordination occurred as follows:

- If *Ready* is true and *Submitted* is false, the human operator module sets one or more of the human action variables to a new input value and sets *Submitted* to true.
- If *Ready* and *Submitted* are true, the human-device interface module responds to the values of the human action variables and sets *Ready* to false.
- If *Ready* is not true and *Submitted* is true, the human operator module sets *Submitted* to false.
- If *Ready* and *Submitted* are both false and the automated system is ready for additional human operator input, the human-device interface module sets *Ready* to true.

3.3 Verification results

Attempts to verify this model using the specifications in Eqs. 1 and 2 resulted in two problems related to the feasibility and usefulness of the verification procedure. First, the SAL–SMC procedure for translating the SAL code into a binary decision diagram (BDD) took excessively long (more than 24 h), a time frame impractical for model debugging. Second, the verification process which followed the construction of the BDD eventually ran out of memory, thus not returning a verification result.

4 Phase 1b: a reduced Baxter Ipump model

As a result of the failed verification of the model produced in Phase 1a, significant revisions were required to make the model more tractable. These are discussed below.

4.1 Representation of numerical values

To reduce the time needed to convert the SAL-code model to a BDD, a number of modifications were made to the model from Phase 1a by representing model constructs in ways more readily processed by the model checker. As such, the modifications discussed here did not ultimately make the BDD representation of the model smaller, but merely expedited its construction.

4.1.1 Redundant representation of values—Two different representations of the values programmed into the pump by the operator were used in the HDI and device automation models. Because the HDI required the human operator to enter values by scrolling through the available values for individual digits, an array of integer digits was appropriate for the HDI model. However, because the system automation was concerned with dynamically checking limits and using entered values to compute other values, a numerical representation of the actual value was more convenient for the automated system model.

This redundancy burdened the BDD translator. This was remedied by eliminating the digit array representations and using functions to enable actions from the human task model to incrementally change individual digits within a value.

4.1.2 Real numbers and integers—In the model produced in Phase 1a, all numerical values were represented as real values with restricted ranges. This was done because most user specified values were either integers or floating point numbers (precise to a single decimal point). No data abstractions were initially considered because the nature of the human task (modeled in Phase 2) required manipulation of values' individual digits. However, representing values this way proved especially challenging for the BDD translator. Thus, all values were modified so that they could be represented as restricted range integers. For integer variables representing floating point numbers, this meant that the model value was ten times the value it represented. This representation allowed the values to still be manipulated at the individual digit level, while making them more readily interpretable by the BDD translator.

4.1.3 Variable ranges—In the Phase 1a model, the upper bound on the range of all value-based variables was set to the theoretical maximum of any value that could be programmed into the pump: 99,999.[3] However, to reduce the amount of work required for the BDD conversion, the range of each numerically valued variable was given a specific upper bound that corresponded to the maximum value it could actually assume in the device.

[3]All lower bounds were set to 0.

4.2 Model reduction

To reduce the size of the model, a variety of elements were removed. In all cases these reductions were meant to reduce the number of state variables in the HDI or device automation models (slicing), or reduce the range of values a variable could assume (data abstraction). Unfortunately, each of these reductions also affected what human tasks could ultimately be modeled and thus verified in subsequent model iterations. All of the following reductions were undertaken:

- In the Phase 1a model, the mission model could generate a prescription from the entire available range of valid prescriptions. This was changed so that fewer prescription options were generated in Phase 1b's mission model: that of programming a prescription with a continuous dosage with two options for bolus delivery (0.0 and 1.0 ml) and two continuous flow rate options (1.0 and 9.0 ml/h). While this significantly reduced the number of model states, it also reduced the number of prescriptions that could be used in verification procedures.

- In the Phase 1a model, the HDI model would allow the operator to select what units to use when entering prescriptions (ml, mg, or μg). Only the ml unit option was included in the Phase 1b model. This reduced the number of interface messages in the model, allowed for the removal of several variables (those related to the unit option selection, and solution concentration specification), and reduced the ranges required for several numerical values related to the prescription. This eliminated the option of including unit selection and concentration specification task behaviors in the model.

- In the Phase 1a model, both the HDI and device automation models encompassed behavior related to the delivery of medication solution during the priming and bolus administration procedures. During priming, the HDI allows the operator to repeatedly instruct the pump to prime until all air has been pushed out of the connected tubing. During bolus administration, the HDI allows the operator to terminate bolus infusion by pressing the stop button twice. This functionality was removed from the Phase 1b models, thus eliminating interface message states and numerical values indicating how much fluid had been delivered in both procedures. This eliminated the possibility of incorporating task behavior related to pump priming and bolus administration in the model.

- The Phase 1a model mimicked the security features found in the original device which required the human operator to unlock and lock the device on startup and enter a security code. This functionality was removed from the Phase 1b model which reduced the number of interface messages in the model and removed the numerical variable (with a 0–999 range) associated with entering the security code. This eliminated the possibility of modeling human task behavior related to unlocking and locking the pump as well as entering the security code in the model.

- In the Phase 1a model, the interface message could automatically transition to being blank: mimicking the actual pump's ability to blank its screen after three seconds of operator inactivity. Because further operator inaction would result in the original device issuing a "left in programming mode" alert, a blank interface message could automatically transition to an alert issuance. This functionality was removed from the Phase 1b model, eliminating several interface messages as well as variables that kept track of the previous interface message. Thus, the option of modeling operator task response to screen blanking and alerts was removed from the model.

While these reductions resulted in the Phase 1b model being much smaller and more manageable than the original, the ability to model some of the task behaviors originally associated with the device had to be sacrificed.

4.3 Results

The Phase 1b model was able to complete the verification procedure with Eq. (1) and produce a counterexample with a search depth of 54 in approximately 5.9 h, with the majority of that time (5.4 h) used for creating the BDD representation [3].[4] Not surprisingly, the model checker ran out of memory when attempting to verify Eq. (2).

5 Phase 1c: a simpler PCA pump model

While the model developed in Phase 1b did produce usable results and has subsequently been used in the verification of additional properties (see [5]), this power came at the expense of a reduction in the scope of the mission model. Since the mission directly influences what human behavior will execute, this limited the human task behavior that could ultimately be verified as part of the system model. Further, the fact that the Phase 1b model was too complex for Eq. (2) to be verified potentially limited any future model development that might add complexity. To remedy these shortcoming, the model produced in Phase 1b was further reduced to one that encompassed the programming of the most basic PCA pump functionality while the ranges of possible values for the remaining mission model variables were expanded to be more realistic.

5.1 Model reduction

To obtain a smaller PCA model, all of the following were removed: the selection of mode and the ability to specify a basal rate, continuous rate, bolus dosage, and fluid volume. As a result, associated interface messages and variables were removed along with the ability to model their programming as part of the human task behavior model. This resulted in a model that only encompassed functionality for programming a PCA dose, programming the delay between PCA doses, turning the pump on and off, and starting and stopping the administration of treatment: functionality compatible with the most basic PCA pump operations (see [1]).

Value ranges were further restricted to reduce the size of the model. Specifically, the upper bound on the acceptable delay between PCA dosages was changed from 240 to 60 minutes. This, coupled with the other reductions, had the added benefit of allowing the number of digits required for the programming of pump values to be reduced to 2 rather than the original 4.

The reductions in other areas allowed the scope of the delays and PCA dosages generated by the mission model to be expanded to a more representative set. For PCA dosages, the full range of values from 0.1 to 9.9 in 0.1 ml increments were supported. For delay between dosages, five options were available: delays of 10, 15, 30, 45, and 60 min.

All pump interface functionality was retained from the previous models. Thus, the unconstrained human task model was unchanged as was the human task and HDI models' communication protocol.

[4]Completed models, SAL outputs, and counterexamples can be found at http://cog.sys.virginia.edu/ISSE2010/.

5.2 Results

The Phase 1c model ran the verification procedure for Eq. (1) (with the eliminated variables removed) in 6 s with a search depth of 22, much faster than the model from Phase 1b. The verification of the specification in Eq. (2) verified to true in 129 s with a search depth of 259 and 78,768,682,750 visited states.

6 Phase 2: incorporating models of human behavior

In the second phase of modeling, we expanded our instantiation of the framework by incorporating a realistic human task behavior model. We therefore replaced the unconstrained human operator in the Phase 1c model with a human task behavior model derived from pump documentation [2] and training materials. This model utilized the EOFM concepts and thus required some additional infrastructure in order to incorporate it into the formal system model. We describe the behaviors that were modeled, how these were translated into the formal model, and report verification results for the produced system model.

6.1 Human task behavior modeling and translation

The pump's materials contained six high-level goal directed behaviors for performing a variety of pump activities relevant to the Phase 1c model as follows:

- Turning on the pump.
- Stopping the infusion of medication.
- Turning off the pump.
- Entering a prescribed value for PCA dosage volumes (in milliliter).
- Entering a prescribed value for the delay between PCA doses (in minutes), and
- Selecting whether to start or review an entered prescription.

The EOFM models describing each of these behaviors are discussed below.

6.1.1 Turning On the pump—The model for turning on the pump is shown in Fig. 3. Here, the EOFM can execute if the interface message indicates that the system is off (*iInterfaceMessage* = SystemOff; a precondition). This high-level activity (aTurnOnPump) is completed by performing the action of pressing the on/off button (hPressOnOff). The ord decomposition operator indicates that all of the decomposed activities or actions must be completed in sequential order. The EOFM has accomplished its goal (a completion condition) when the interface message indicates that the pump is no longer off (*iInterfaceMessage*/ = SystemOff).

6.1.2 Stopping infusion—Infusion of medication can be stopped (Fig. 4) if the interface indicates that treatment is administering (*iInterfaceMessage* = TreatmentAdministering). This is accomplished by pressing the stop button (hPressStop) twice in quick succession with no other human inputs occurring in between. The process has completed when the interface indicates that treatment is not administering (*iInterfaceMessage* / = TreatmentAdministering).

6.1.3 Turning Off the pump—The model for turning off the pump (Fig. 5) is relevant if the interface message indicates that the system is not off (*iInterfaceMessage* / = SystemOff). The pump is turned off by performing two lower level activities in sequential order: stopping infusion (aStopInfusion; explained above) and pressing the keys necessary to turn off the pump (aPress-KeysToTurnOffPump). This latter activity is completed by pressing the on/off button (hPressOnOff) twice in sequence. The entire process of turning off the pump completes when the interface indicates that the pump is off (*iInterfaceMessage* = SystemOff).

6.1.4 Programming a value into the pump—The values for PCA dosage volume and delay between dosages can be programmed into the pump using an EOFM patterned after Fig. 6. Thus, for a given value *X*, the corresponding EOFM becomes relevant when the interface for setting that value is displayed (*iInterfaceMessage* = Set*X*). This is achieved by sequentially executing two sub-activities: changing the displayed value (aChange*X* Value) and accepting the displayed value (aAccept). The activity for changing the displayed value can execute, and will repeatedly execute, if the displayed value is not equal to the prescribed value (*iCurrentValue* / = *iPrescribedX*). The value is changed by executing one or more (denoted by the or_seq decomposition operator) of the following sub-activities: changing the digit currently pointed to by the cursor (aChangeDigit: completed by pressing the up key (hPressUp)), moving the cursor to a different digit (aNextDigit: completed by pressing only one of (the xor decomposition operator) the left (hPress-Left) or right (hPressRight) buttons), or setting the displayed value to zero (aClearValue: completed by pressing the clear button(hPressClear)). The process of changing the displayed value completes when the displayed value matches the prescribed value (*iCurrentValue* = *iPrescribedX*). The displayed value is accepted by pressing the enter key. The entire process ends when the interface is no longer in the state for accepting *X*.

6.1.5 Starting or reviewing a prescription—After a prescription has been programmed the human operator is given the option to start the administration of that prescription or to review it (where the operator works through the programming procedure a second time with the previously programmed options displayed at each step). The EOFM for performing this (Fig. 7) becomes relevant at this point (*iInterfaceMessage* = StartBeginsRx). It is completed by performing only one of two activities: selecting the option to start treatment (aStartRx—performed by pressing the start button (hPressStart)) or selecting the review option (aReviewRx—performed by pressing the enter button (hPressEnter)).

6.2 EOFM translation

The EOFMs representing the human task model were translated into a SAL code module. This translation was accomplished by creating a variable for each activity or action node in each EOFM, each of which could assume one of three enumerated values describing its execution state: ready, executing, or done. Thus, in addition to handling the transitional logic for the coordination protocol, this module handled the transition logic for allowing the variables representing activity and action nodes to transition between these three values. All activity and action variables start in the ready state. They can transition between execution states based on the execution state of their children, parent, and siblings in the EOFM structure; the evaluation of their conditions; and their position within the EOFM hierarchy.

While the resulting human operator module and its associated unconstrained operator model both had the same inputs and outputs, the logic associated with traversal of the human task structures required 48 additional variables in the human task behavior model.

6.3 Results

The specification in Eq. (1) verified (produced the expected counterexample) in 57 s with a search depth of 42. The specification in Eq. (2) verified to true in 10.6 h with a search depth of 437 and 1,534,862,538 visited states.

7 Discussion

This work has shown that it is possible for human–automation interaction to be evaluated using the architecture in Fig. 1. However, this came as a result of tradeoffs between the goals the architecture is designed to support:

1. Model constructs need to be intuitive to human factors engineers who will be building and evaluating many of the models;
2. the sub-models should be decoupled and modular (as in Fig. 1) in order to allow for interchangeability of alternative sub-models; and
3. the constructed models need to be capable of being verified in a reasonable amount of time.

We discuss how each of these goals was impacted and how related issues might be addressed.

7.1 Goals 1: model intuitiveness

Many of the model revisions were associated with representing model constructs in ways that were more readily interpretable by the model checker rather than the human factors engineer. These primarily took the form of converting floating point and digit array values into integers in Phase 1b. Further, the extensive model reductions that were undertaken in Phase 1c would be very cumbersome for a human factors engineer.

There are two potential ways to address this issue. One solution would be to improve the model checkers themselves. Given that the modifications would not actually change the number of reachable states in the system, this would suggest that the model checker need only optimize the BDD conversion algorithms.

Alternatively, additional modeling tools could be used to help mitigate the situation. Such tools could allow human factors engineers to construct or import HDI prototypes, and translate them into model checker code. This would allow the unintuitive representations necessary for ensuring a model’s efficient processing by the model checker to be removed from the modeler’s view.

7.2 Goal 2: decoupling of architecture sub-models

Because the protocol used to coordinate human actions between the HDI and human task models (discussed for Phase 1a and used in all models produced in all subsequent phases) assumes a particular relationship between variables shared by these models, they are tightly coupled. Unless a model checker can be made to support both asynchronous and synchronous relationships between models more elegantly, this coordination infrastructure cannot be eliminated.

However, a solution may be found in an additional level of abstraction. A toolset for translating a HDI prototype into model checking code, could handle the construction of the coordination protocol, making this process effectively invisible to the modeler. Such a process could also allow for more efficient means of coordinating the HDI and human task models: one that might not require the use of separate models in the actual model checker code.

While the extensive model reductions from Phase 1 greatly diminished the fidelity with which the model represented the actual PCA pump, this provides some advantages. Since the model from Phase 2 does not suffer from the memory usage problems encountered in Phase 1, this opens the door to the addition of other model constructs to be added allowing for a more complete system analysis. Future work can expand the model developed in Phase 2 with environmental and device automation models that are compatible with the formal PCA pump reference model described in [1].

7.3 Goal 3: model verifiability

We are predominantly concerned with exploring how formal methods can be used to provide insights into human factors and systems engineering concerns. If our goal was to formally verify properties of the Baxter Ipump, the modeling compromises we made in order to obtain a verifiable model might necessitate a change in modeling philosophy or verification approach.

There are many barriers to the verifiability of models of realistic systems. These include large numbers of parallel processes, large ranges of discrete valued variables, and non-discretely valued variables. The modeling efforts described here were so challenging because the target system was dependent on a large number of user specified numerical values, all of which had very large acceptable ranges. This resulted in the scope of the model being reduced to the point where it could no longer be used for verifying all of the original human operator task behaviors: with the model produced in Phase 1b making minor compromises and the model produced in Phase 1c only allowing for behaviors associated with basic PCA pump functionality.

As was demonstrated in Phase 2, the verifiability of the model actually increased with the inclusion of the human task behavior as indicated by the 98% reduction in the reported state space from the Phase 1c to the Phase 2 model. However, this came at the expense of the verification process taking 284 times as long. Thus, in a context where verification time is less important than the size of the model’s state space, the inclusion of the human task behavior model may generally prove to be advantageous in the formal verification of systems that have a human–automation interaction component, irrespective of whether the human behavior is of concern in the verification process. Future efforts should investigate the different factors that affect this tradeoff.

Even exploiting this advantage, the relative simplicity of the device that was modeled in this work makes it clear that there are many systems that depend of human–automation interaction that would be even more challenging to verify, if not impossible, using these techniques. While the use of bounded model checkers may provide some verification capabilities for certain systems, there is little that can be done without either using additional abstraction techniques or future advances in model checking technology and computation power.

It is common practice in the formal methods community to use more advanced forms of data abstraction than those employed in this work to mitigate the complexity of variables with large value ranges (an overview of these methods can be found in [20]). Because the nature of the modeled human task behavior in this work was concerned with the digit level editing of the data values, such abstractions were not appropriate for this particular endeavor. Additionally, automatic predicate abstraction techniques like those used in counterexample-guided abstraction refinement [6] could potentially alleviate some of the complexity problems encountered in this work without requiring changes to the models themselves. Future work should investigate how these different abstraction techniques could be used when modeling systems that depend on human-automation interaction in ways that are intuitive to human factors engineers.

It is clear that the multiple, large-value-ranged variables were the source of most of the model complexity problems in the pump example, as shown in the drastic decrease in verification performance time between the models produced in Phases 1b and 1c. Thus, had the target system been more concerned with procedural behaviors and less on the interrelationships between numerical values, the system model would have been much more tractable. Future work should identify additional properties of systems dependent on human–automation interaction that lend themselves to being modeled and verified using the framework discussed here.

Finally, some of the performance issues we encountered can be attributed to our use of SAL. For example, model checkers such as SPIN [15] do not perform the lengthy process of constructing the BDD representation before starting the checking process. Future work should investigate which model checker is best suited for evaluating human–automation interaction.

8 Conclusion

The work presented here has shown that it is possible to construct models of human–automation interaction as part of a larger system for use in formal verification processes while adhering to some of the architectural goals in Fig. 1. It has also shown that the incorporation of human task behavior models into system models may help alleviate the state explosion problem in some systems that depend on human–automation interaction. However, this success was the result of a number of compromises that produced a model that was not as representative, understandable, or modular as desired. Thus, in order for formal methods to become more useful for the HFE community, the verification technology will need to be able to support a more diverse set of systems. Further, new modeling tools may be required to support representations that human factors engineers use. These advances will ultimately allow formal methods to become a more useful tool for human factors engineers working with safety critical systems.

Acknowledgments

The research described was supported in part by Grant Number T15LM009462 from the National Library of Medicine and Research Grant Agreement UVA-03-01, sub-award 2623-VA from the National Institute of Aerospace (NIA). The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIA, NASA, the National Library of Medicine, or the National Institutes of Health. The authors would like to thank Radu I. Siminiceanu of the NIA and Ben Di Vito of the NASA Langley Research Center for their technical help. They would like to thank Diane Haddon, John Knapp, Paul Merrel, Kathryn McGough, and Sherry Wood of the University of Virginia Health System for describing the functionality of the Baxter Ipump and for providing documentation, training materials, and device access.

References

1. Arney, D.; Jetley, R.; Jones, P.; Lee, I.; Sokolsky, O. Formal methods based development of a PCA infusion pump reference model: generic infusion pump (GIP) project; Proceedings of the 2007 joint workshop on high confidence medical devices, software, and systems and medical device plug-and-play interoperability; Washington, DC: IEEE Computer Society; 2007. p. 23-33.
2. Baxter Health Care Corporation. Ipump pain management system operator’s manual. McGaw Park: Baxter Heath Care Corporation; 1995.
3. Bolton, ML.; Bass, EJ. Building a formal model of a human-interactive system: insights into the integration of formal methods and human factors engineering; Proceedings of the first NASA formal methods symposium; Moffett Field: NASA Ames Research Center; 2009. p. 6-15.
4. Bolton, ML.; Bass, EJ. Enhanced operator function model: a generic human task behavior modeling language; Proceedings of the IEEE international conference on systems, man, and cybernetics; Piscataway: IEEE; 2009. p. 2983-2990.

5. Bolton, ML.; Bass, EJ. A method for the formal verification of human-interactive systems; Proceedings of the 53rd annual meeting of the human factors and ergonomics society; Santa Monica: Human Factors and Ergonomics Society; 2009. p. 764-768.
6. Clarke E, Grumberg O, Jha S, Lu Y, Veith H. Counterexample-guided abstraction refinement for symbolic model checking. J ACM. 2003; 50(5):752–794.
7. Crow, J.; Javaux, D.; Rushby, J. Models and mechanized methods that integrate human factors into automation design; Proceedings of the 2000 international conference on human-computer interaction in aeronautics; Menlo Park: Association for the Advancement of Artificial Intelligence; 2000. p. 163-168.
8. Curzon P, Ruksenas R, Blandford A. An approach to formal verification of human–computer interaction. Formal Asp Comput. 2007; 19(4):513–550.
9. De Moura, L.; Owre, S.; Shankar, N. Technical report. Menlo Park: Computer Science Laboratory, SRI International; 2003. The SAL language manual.
10. Degani, A. PhD thesis. Atlanta: Georgia Institute of Technology; 1996. Modeling human–machine systems: on modes, error, and patterns of interaction.
11. Degani, A.; Kirlik, A. Modes in human–automation interaction: Initial observations about a modeling approach; Proceedings of the IEEE international conference on systems, man and cybernetics; Piscataway: IEEE; 1995. p. 3443-3450.
12. Fields, RE. PhD thesis. York: University of York; 2001. Analysis of erroneous actions in the design of critical systems.
13. Heymann M, Degani A. Formal analysis and automatic generation of user interfaces: approach, methodology, and an algorithm. Hum Factors. 2007; 49(2):311–330. [PubMed: 17447671]
14. Heymann, M.; Degani, A.; Barshi, I. Generating procedures and recovery sequences: a formal approach; Proceedings of the 14th international symposium on aviation psychology; Dayton: Association for Aviation Psychology; 2007. p. 252-257.
15. Holzmann, GJ. The spinmodel checker, primer and reference manual. Reading: Addison-Wesley; 2003.
16. Javaux D. A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system. Reliab Eng Syst Saf. 2002; 75(2):147–165.
17. Kirwan, B.; Ainsworth, LK. A guide to task analysis. Philidelphia: Taylor and Francis; 1992.
18. Kohn, LT.; Corrigan, J.; Donaldson, MS. To err is human: building a safer health system. Washington: National Academy Press; 2000.
19. Krey, N. 2007 Nall report: accident trends and factors for 2006. Technical report. 2007. http://download.aopa.org/epilot/2007/07nall.pdf
20. Mansouri-Samani, M.; Pasareanu, CS.; Penix, JJ.; Mehlitz, PC.; O'Malley, O.; Visser, WC.; Brat, GP.; Markosian, LZ.; Pressburger, TT. Technical report. Moffett Field: Intelligent Systems Division, NASA Ames Research Center; 2007. Program model checking: a practitioner's guide.
21. Mitchell CM, Miller RA. A discrete control model of operator function: a methodology for information dislay design. IEEE Trans Syst Man Cybern A Syst Hum. 1986; 16(3):343–357.
22. Perrow, C. Normal accidents. New York: Basic Books; 1984.
23. Rushby J. Using model checking to help discover mode confusions and other automation surprises. Reliab Eng Syst Saf. 2002; 75(2):167–177.
24. Schraagen, JM.; Chipman, SF.; Shalin, VL. Cognitive task analysis. Mahwah: Lawrence Erlbaum Associates; 2000.
25. Stanton, N. Human factors methods: a practical guide for engineering and design. Brookfield: Ashgate Publishing; 2005.
26. Thurman, DA.; Chappell, AR.; Mitchell, CM. An enhanced architecture for OFMspert: a domain-independent system for intent inferencing; Proceedings of the IEEE international conference on systems, man, and cybernetics; Piscataway: IEEE; 1998. p. 3443-3450.
27. Vicente, KJ. Cognitive work analysis: toward safe, productive, and healthy computer-based work. Mahwah: Lawrence Erlbaum Associates; 1999.
28. Wells, AT.; Rodrigues, CC. Commercial aviation safety. 4th edn. New York: McGraw-Hill; 2004.

29. Wickens, CD.; Lee, J.; Liu, YD.; Gordon-Becker, S. Introduction to human factors engineering. Upper Saddle River: Prentice-Hall; 2003.

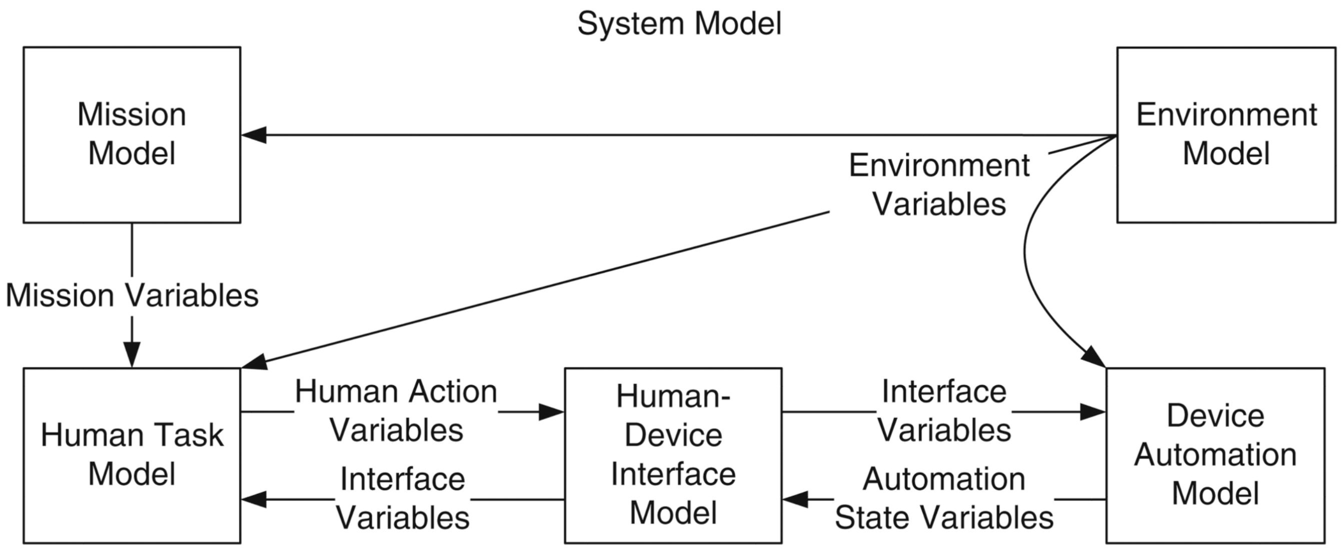

Fig. 1.
Framework for the formal modeling of human–automation interaction. *Arrows* between models represent variables that are shared between models. The direction of the arrow indicates whether the represented variables are treated as inputs or output. If the arrow is sourced from a model, the represented variables are outputs of that model. If the arrow terminates at a model, the represented variables are inputs to that model

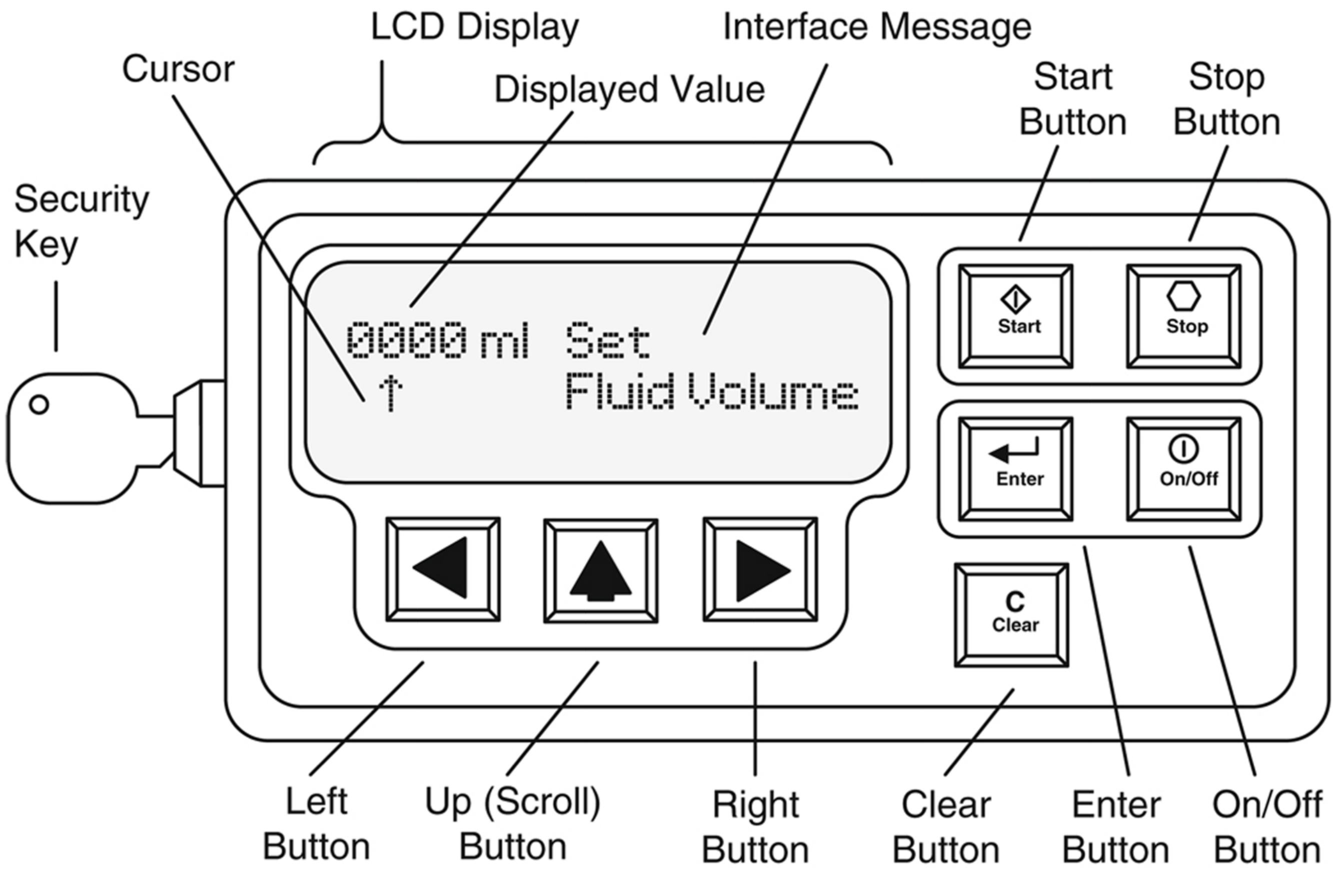

Fig. 2.
A simplified representation of the Baxter Ipump's human-device interface. Note that the actual pump contains additional controls and information conveyances

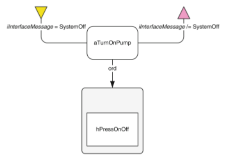

Fig. 3.
The EOFM graphical representation for turning on the pump

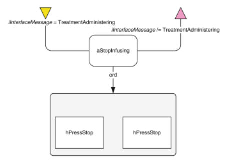
iInterfaceMessage = TreatmentAdministering
iInterfaceMessage /= TreatmentAdministering
aStopInfusing
ord
hPressStop
hPressStop

Fig. 4.

The EOFM graphical representation for stopping infusion

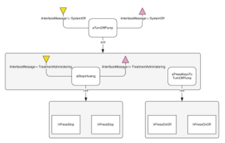
iInterfaceMessage != SystemOff
iInterfaceMessage = SystemOff
aTurnOffPump
ord
iInterfaceMessage = TreatmentAdministering
iInterfaceMessage != TreatmentAdministering
aStopInfusing
aPressKeysTo TurnOffPump
ord
ord
hPressStop
hPressStop
hPressOnOff
hPressOnOff

Fig. 5.
The EOFM graphical representation for turning off the pump

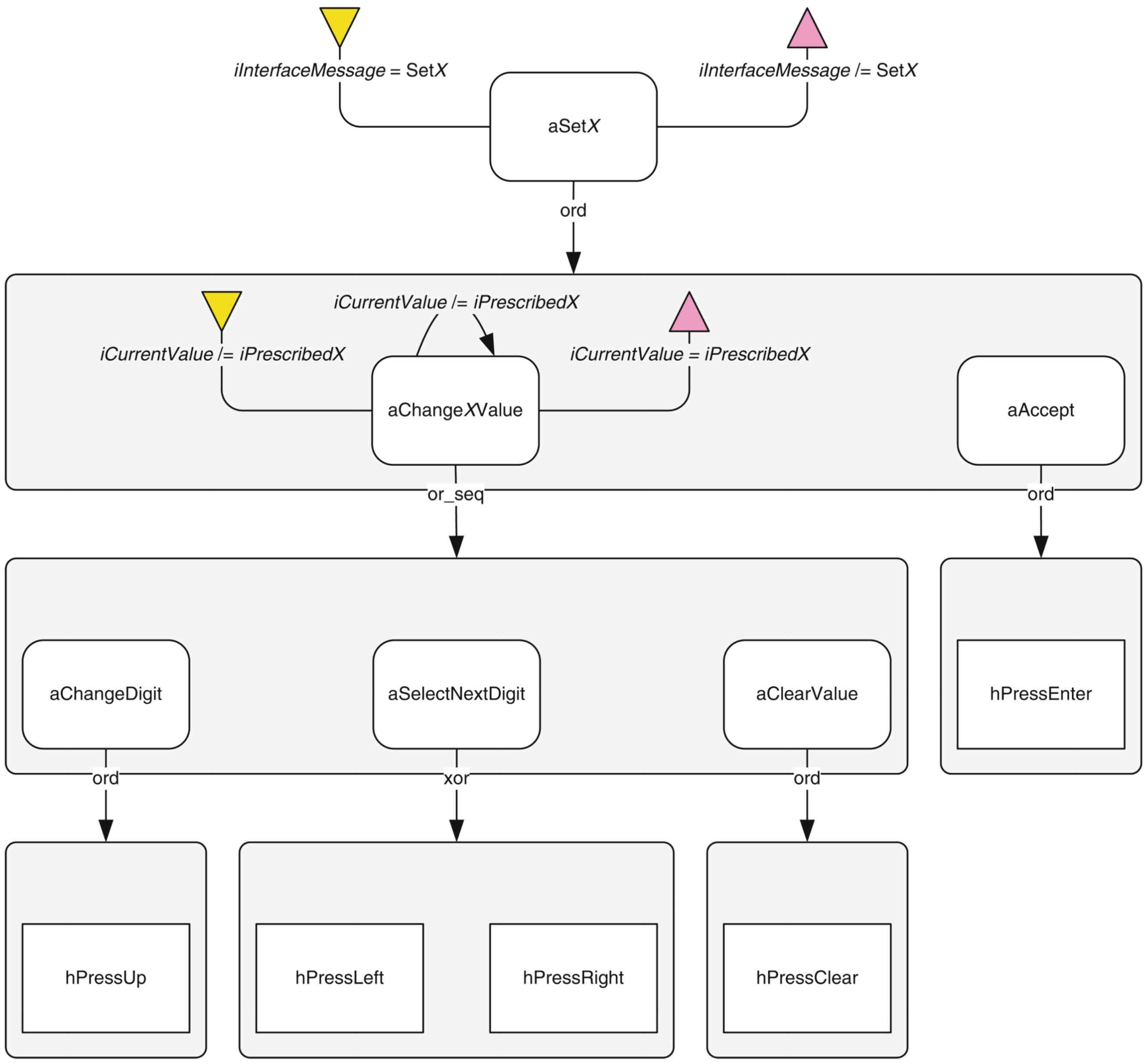

Fig. 6.
The EOFM graphical representation of the pattern for programming a value *X* into the pump

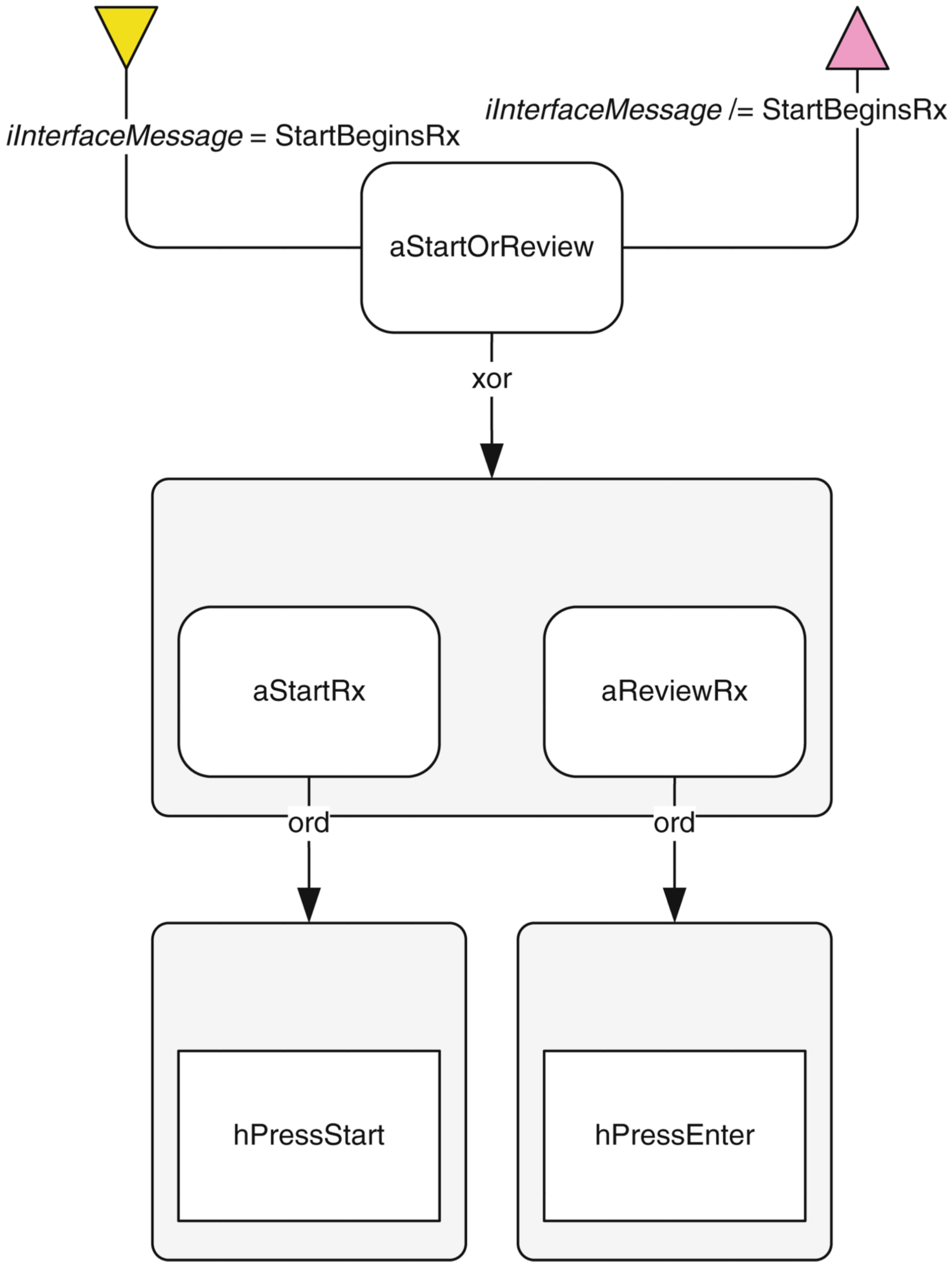
iInterfaceMessage = StartBeginsRx
iInterfaceMessage /= StartBeginsRx
aStartOrReview
xor
aStartRx
aReviewRx
ord
ord
hPressStart
hPressEnter

Fig. 7.
The EOFM for choosing to start or review a prescription