MODELING AND FORMAL ANALYSIS OF HUMAN-MACHINE INTERACTION

Asaf Degani
General Motors R&D, Advanced Technical Center Israel

Michael Heymann
Dept. of Computer Science, Israel Institute of Technology

Michael Shafto
Intelligent Systems Division, NASA Ames Research Center, California

Abstract

We describe a formal, model-based, approach for the analysis and design of human-automation interaction. The approach is guided by two primary objectives: first, that the interaction be correct in the sense that it will enable the user to operate the system reliably; second, that the interaction be as simple as possible while maintaining all the required operational flexibility. With these objectives we first discuss a methodology for verification of user-interface correctness, thereby addressing the issue of operational reliability, and then present an algorithmic approach for designing simple and succinct interfaces.

Introduction

There are numerous ways to model human interaction with machines. Some models focus on the perceptual and cognitive aspects of the interaction, such as human memory, response capabilities, attention span, decision-making, and action-selection (e.g., see Endsley, Wickens, Salvucci, this *Handbook*). The aim of these models is to describe and analyze human abilities and limitations in the context of task execution. These models are also aimed at understanding users' skill acquisition (e.g., see Ritter et al., this *Handbook*). An important objective of such models is to predict conditions under which human interaction may become unduly stressful and error-prone.

Traditionally, faulty interactions of users with machines, sometimes leading to catastrophic results, have been attributed to machine functionality failures or human error, with the latter sometimes blamed on inadequate interface design. However, such faulty interactions can also occur due to unpredictable responses of the machine to user interaction, leading to erroneous interpretation of the machine's behavior. Such ambiguities arise due to design errors, such that the information supplied to the user about the machine's status is insufficient, untimely, or inaccurate. Ambiguities may also

arise when the user is offered an inadequate set of possible actions that will not enable the achievement of operational goals or the desired level of performance.

We illustrate the subtle issue of unpredictable machine responses with the following example: An airliner is flying at 8,000 feet under autopilot control. The crew receives an air traffic control directive to climb and level off at 10,000 feet. The pilot enters the 10,000-foot altitude constraint into the autopilot, engages a mode called "VERTICAL SPEED," and then selects a rate of climb (e.g., 2,000 feet/min). When the aircraft reaches an altitude of 9,000 feet, air traffic control directs the crew to descend back to 8,000 feet. In response, the pilot enters the new altitude of 8,000 feet into the autopilot. Normally, the pilot would expect the aircraft to descend and level off at 8,000 feet. However, it is also possible that the aircraft continues climbing indefinitely (with an eventual stall), unless some control action is taken by the crew (NTSB, 1980).

In this situation, under identical interface annunciations and the same pilot input (i.e., entering the new 8,000 feet altitude setting when the aircraft is at 9,000 feet), two different outcomes might occur with the crew being unable to predict which. The problem here is not that the autopilot behaves ambiguously: The autopilot, in fact, is fully deterministic. It is the interface that does not provide sufficient information to resolve the situation. What happens is that if the newly entered altitude setting (8,000 feet) is above a certain reference value, the aircraft will descend and level off as expected, but if the newly entered altitude is below the reference value, the aircraft will continue climbing. Yet, this reference altitude value (which changes as a function of the aircraft's speed and altitude) is not presented on the interface. Such an interface is defined as "incorrect" because the consequences of user interaction with the machine cannot be predicted reliably.

In this Chapter we address problem of insuring correct human-machine interaction as measured by the ability of the user to perform reliably a set of specified tasks. That is, making sure that the machine interface has been properly designed to enable the user to achieve the required operating tasks. We consider such interfaces to be "correct".

Hence the first main topic of this chapter deals with a formal, model based, methodology for verifying interface correctness and for detecting design deficiencies. The models used in this formal verification are derived from the behavioral description of the machine and the operating specifications given in the user-manual and reflected in the interface. These models are then systematically tested for correct correspondence and compatibility. Deficiencies are detected and classified.

The second topic concerns design. Traditionally, the practice of designing user interfaces has been based on a bottom-up approach. The designer starts with the operating specifications of the machine, the operational constraints, and other desired information regarding the system. The interface is directly designed to accommodate these operational specifications providing the user with suitable inputs to the machine and information about the machine's behavior. For example, in a motor vehicle, the interface includes input devices such as steering wheel, gas pedal, gearshift and brakes, as well as information about the car's speed, distance of travel, and quantity of fuel. The interface is then further enhanced by features that the designer deems necessary for the user to have in order to operate the machine efficiently. For example, whether the engine's temperature is normal or dangerously high. Such added information is usually based on the designer's knowledge about the machine. To insure that the user interface is ultimately adequate, designers rely on reviews and inspections, simulations and extensive testing. This approach is costly and not always sufficient, especially in automated control systems that are quite large and exhibit complex behaviors and interactions.

Elements of Human-Machine Interaction

In the interaction between the user and the machine, four elements play crucial roles. These are (1) the machine's behavior; (2) the user's task specification; (3) the user interface; and (4) the model that the user has about the machine's behavior (as described in the user-manual). These four elements must correspond suitably to insure correct and reliable user-machine interaction.

(1) We consider machines that can be modeled as state transition systems. There are two classes of transitions: those that are manually triggered by the user, and those that are automatically triggered (either by the machine's internal dynamics or by the environment).

(2) Formally, a task consists of driving the system to a specified set of states or to a prescribed sequence(s) of such sets. Since it is not important which specific state in such a set the machine visits, the user is not required to distinguish among them provided the task can be executed correctly and reliably. Accordingly, we partition the machine's state-space into disjoint clusters that we shall call "specification classes." A specification class is then a set of internal states that, according to the design specifications, need not be distinguished by the user. The partition is typically performed based on task analysis and judgments of expert users and developers. The most common task requirement is for the user to track these specification classes, unambiguously.

(3) The interface consists of input devices such as push buttons, rotary switches, and dials through which the user interacts with the machine, as well as indications through which the user obtains information about the machine's state.

(4) The user model constitutes a formal description of possible user actions and machine responses as described by the designer and represented in the operation manual. This information is commonly provided in the form of fragmented statements such as "When the machine is in mode A and button x is pushed, the machine transitions to mode B." The user-model is the assembly of these fragments into a complete state transition system.

To illustrate the relations between these four elements, consider the simple machine described in Figure 1(a).

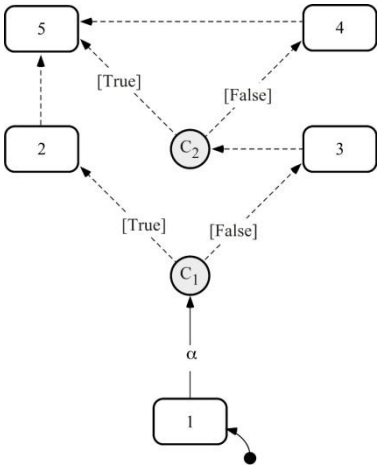


Figure 1(a): Machine model.

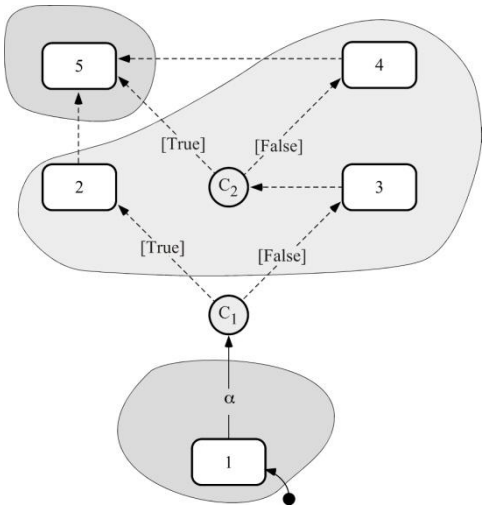


Figure 1(b): Machine mode and specifications classes (for task TS-1).

The machine starts at state 1, and upon execution (by the user) of event α, transitions to either state 2 or state 3, depending whether condition C_1 is true or false. The dashed arrows represent transitions that are automatically triggered by the machine (i.e., without direct user's intervention). Thus, if state 2 is reached, the system moves automatically to state 5, while if it reaches state 3, it moves to either state 5 or to state 4 depending on whether condition C_2 is true or false.

Suppose that the first task specification (call it TS-1) is to drive the system to state 5. In this case, regardless of whether conditions C_1 and/or C_2 are true or false, state 5 will be the guaranteed

outcome of executing event α. Thus, it is inconsequential for the user to know which path the system takes from state 1 to state 5 after executing event α. Therefore, states 2, 3, and 4 can be grouped into a single specification class for task (TS-1) as depicted in Figure 1(b). And since this specification class is actually superfluous to the user in this case, these three states can be omitted altogether, resulting in the simplified user model of 1(c).

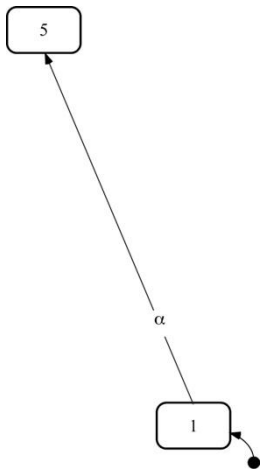


Figure 1(c): User model (for task TS-1)

Suppose we now have another task specification. This new task specification (TS-2) is to drive the system from state 1 to state 5 by way of state 3. For this task, it is necessary that condition C_1 be false. Four specification classes are defined in Figure 1(d). States 3 and 4 share the same specification class because it does not matter whether we reach state 5 directly from state 3 or via 3 and then 4. To correctly and reliably execute the task the user only needs to know whether condition C_1 is true or false before executing α. Since the specification class containing state 2 is superfluous to the user in this case, this state can be omitted from the user model (Figure 1(e)).

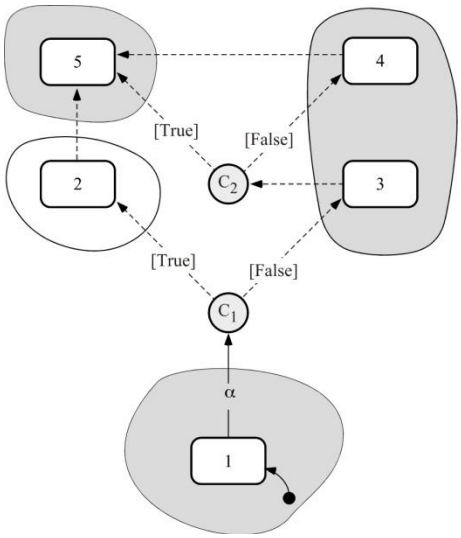


Figure 1(d): Machine mode and specifications classes (for task TS-2).

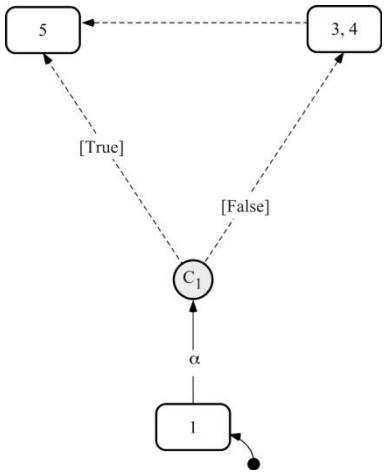


Figure 1(e): User model (for task TS-2).

In general, as can also be seen in the above example, there are events that are triggered by the user while other events are triggered by the machine (through a variety of internal mechanisms). While user triggered events must always be presented in the user model, machine triggered events may or may not be monitored by the user provided the task specification can be met. Such events can be presented explicitly in the user model, omitted from the user model entirely, or grouped together (by presenting several events as a single event in the user model).

Verification of the User Model

We are given a machine, an interface, and a user-model as proposed by the engineering design team. The interface and the associated user-model constitute a reduced description of the actual behaviors that take place in the machine. We are asked to verify whether the proposed interface and user model are correct for the task(s) that the user is expected to perform with this machine. This is the verification problem (Degani & Heymann, 2002).

To get an insight into this problem consider the machine model presented in Figure 2(a). It consists of 5 states and a set of transitions ($\alpha, \beta, \gamma, \delta,$ and μ) that are monitored by the user. The user's task is to monitor the machine and predict its entry into state 4, which is illegal. Thus, there are two specification classes here: the legal class (containing states 1, 2, 3,5) and the illegal class (state 4).

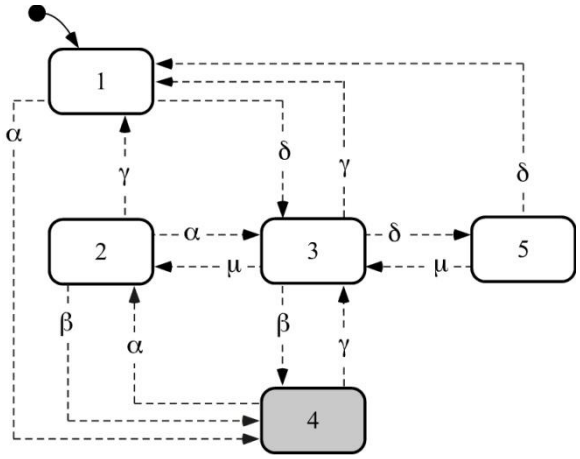


Figure 2(a): Machine model.

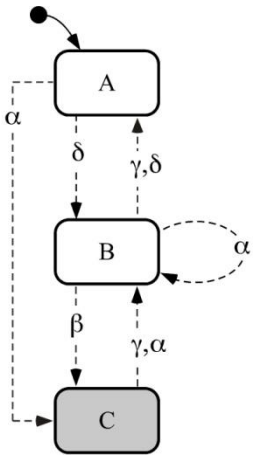


Figure 2(b): User model

Let us suppose that we are provided with a proposed user model that is intended to enable the user to monitor the machine as required by the task specification. This user model contains three states, *A, B,* (corresponding to the legal specification class) and *C,* which corresponds to the illegal state 4 (see Figure 2(b)). Thus when monitoring the machine through this model the user would expect the user model enters to enter state C if and only if the machine enters state 4.

The user monitors the progress of the machine through the interface. The interface displays the three user model states (A, B, and C) and indicates which one is active. All events of the original machine appear in the user-model, except for event μ, which the user presumably need not monitor to achieve the task specification. Since the interface is based on the user-model's description of the machine behavior, there is no difference, for the purpose of analysis, between the user model and the (graphical aspects of the) interface. We therefore only employ the user model and the machine model in the verification process.

Verification Procedure

Let us try to see how one could verify whether the proposed user-model (and the interface embedded in it) is correct for the task of monitoring the machine and predicting entry into the illegal state 4. The user tracks the progress of the machine via the interface with the aid of the user-model. The machine starts at state 1 and the proposed user-model for the machine starts at state A. The machine can respond to event α and move to state 4, or to event δ and move to state 3. In response to the event α the user model transitions to state C while in response to δ it moves to B. Thus, when the user interacts with the machine, the machine model and user model run in parallel (and synchrony). This parallel run can be figuratively thought of as the operation of a "composite

machine" in which each composite-state represents a pair of states (one from the machine and one from the user model). Thus the composite machine starts at the pair A1 and in response to event α moves to C4 and in response to δ moves to B3. The composite machine is computed through a concurrent execution of the two respective models (Figure 2(c) shows a relevant fragment of the full composite).

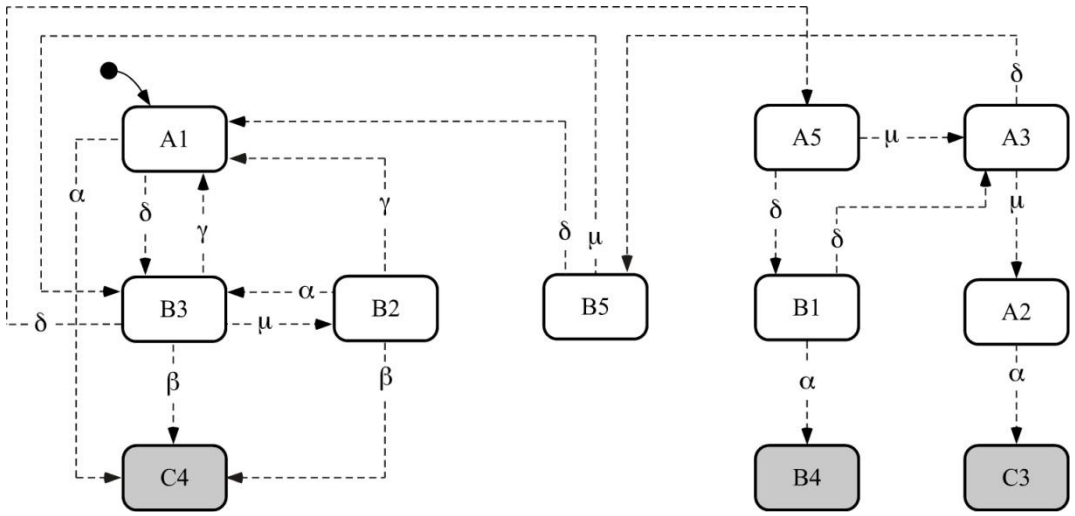


Figure 2(c): Composite model (fragment).

The basis of the verification procedure rests on a careful analysis of the composite machine. We wish to verify whether the user-model enters the designated illegal state C if and only if the machine model enters the illegal state 4. Thus, in the composite machine, when a state-pair is entered, the component states must both be either legal or both illegal. A violation of this requirement implies that the user model is incorrect (for the specified task). Mixed state pairs, where one is legal while the other is not, are called "error states".

As can be seen in the Figure, the composite model can enter not only the state C4 where both elements are illegal, but also states B4 and C3 where one component state is legal while the other is not. Both B4 and C3 are thus error-states indicating that the user model is incorrect.

Event Discrepancy

The composite model above can reach states A3 and A2 which exhibit other types of inadequacies: At state 3 the machine model can respond to events β, γ, δ and μ, while at state 2 the machine can respond to events α, β, γ and μ (recall that event μ has been deliberately omitted from the interface). In contrast, the user model, in state A, can respond only to events α and δ. Thus there exist

discrepancies between the response capabilities of the machine model and the user model at these composite states. For example, at state 3 the machine model responds to event γ, which the user model does not indicate as available in A. Similarly, event α which the user model exhibits as possible at state A cannot occur at all when the machine is at state 3. In many machines the user does not just monitor events but also operates the machine by triggering event transitions. In these situations event discrepancies might mean that the user may try to trigger events that are inoperative at the current machine state or alternatively be unaware of operability of certain events. In summary, event discrepancies do not necessarily present practical difficulties, but they do need to be considered and dealt with on a case-by-case basis. Correspondingly, events can be added to or omitted from the interface as needed.

An Alternative User Model

Figure 2(d) presents an alternative user model for the machine model and task specification of monitoring the machine and predict its entry into the illegal state 4. It contains three states as before, but event μ is not omitted and events γ and δ are grouped as a single event θ (meaning that the user model does not distinguish between them and views them as equivalent).

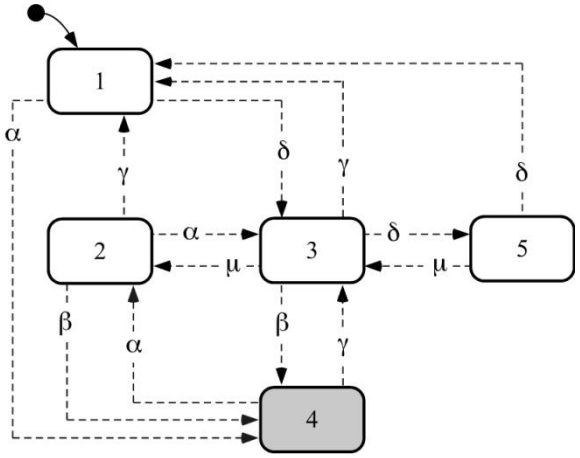


Figure 2(a): Machine model.

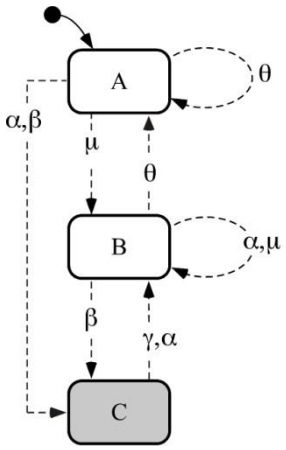


Figure 2(d): Alternative user model

The full composite model is presented in Figure 2(e). The composite contains no error states and since there are no user-triggered event transitions we are not overly concerned about event discrepancies in this particular case. We therefore conclude that this new user-model is correct for this monitoring task.

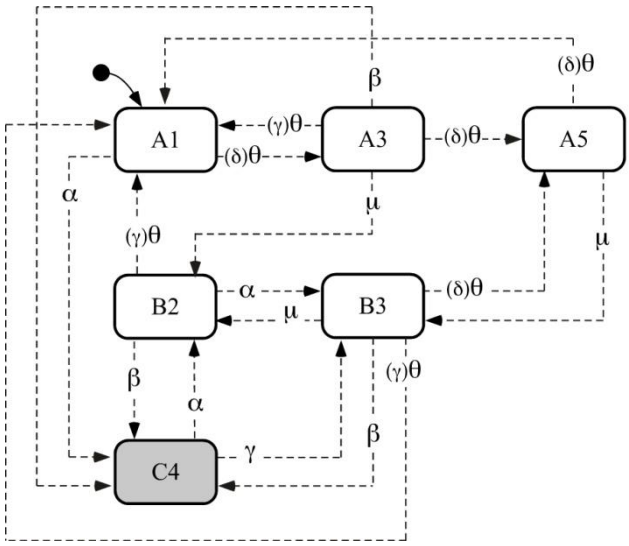


Figure 2(e). Composite model for the alternative user model.

Note that in the above example there existed only two specification classes: the set of legal (or wanted) states (1,2,3, and 5), and the set of illegal (or unwanted) states (4). In general, there will be more than two specification classes for a task, in which case error states are any composite model pair where the two component states belong to distinct specification classes.

Verification of An Autopilot Behavior

In this section we shall briefly demonstrate the usefulness of the methodology by detailing the incorrect interface problem described in the introduction. We focus our attention on the vertical flight modes of the aircraft: the "ALTITUDE HOLD" mode, in which a specified altitude value is maintained constant, and the "VERTICAL SPEED" mode which controls the climb or descent rates.

Machine Model

Figure 3(a) shows a small fragment of the transitions among the vertical-flight modes. The vertical speed mode is either constrained by a target altitude (V/S to altitude setting), in which case a target altitude is armed for eventual capture, or it may be unconstrained (V/S unconstrained), in which case no target altitude is armed.

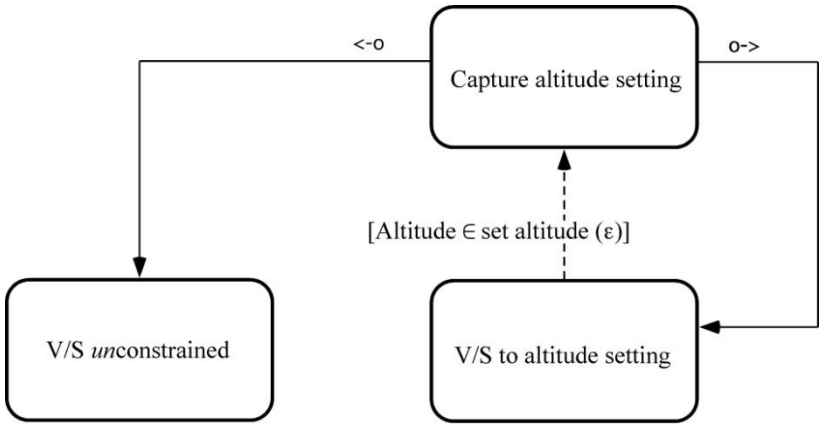


Figure 3(a). Machine model of vertical modes.

Each of the modes can be thought of as a distinct aircraft activity, which is fully defined in the autopilot and has an associated set of parameters. For example, the “V/S to altitude setting” represents a climb or descent activity, and is parameterized by the value of the vertical speed setting and the target-altitude setting. The “V/S unconstrained” activity is only parameterized by the value of the vertical speed setting.

Some of the transitions among the modes are triggered by the internal dynamics of the system. When the aircraft altitude reaches a value that satisfies a certain dynamic condition, denoted formally as [Alt ∈ set altitude (ε)], an automatic transition to the activity "Capture altitude setting" takes place. The "Capture altitude setting" activity maneuvers the aircraft to the target altitude.

The pilot can change the altitude setting at any time. There is a qualitative difference in the autopilot’s behavior depending on whether the altitude is changed to a value ahead or behind a specific critical altitude. Here by ahead we mean in a temporal sense, that is, "higher than" the critical altitude when climbing, and "lower than” the critical altitude when descending. (Similarly, behind means "lower than" when climbing, and "higher than" when descending). These altitude-setting changes are shown in the machine model of Figure 3(a) by the transition labels “o->” for ahead and “<-o” for behind. Thus, when the autopilot is in the activity “V/S to altitude setting,” changing the altitude setting to a value behind the critical altitude triggers a transition to “V/S unconstrained.” In this activity, the aircraft continues the current climb/descent without any effective altitude constraint. Changing the altitude setting to a value ahead of the critical altitude triggers a transition to the “V/S to altitude setting” activity, parameterized by the newly set altitude.

User Model

The user model is derived from the user manual provided to the pilot. The description of the interface as relates to the vertical speed activities discussed above is shown in Figure 3(b). It shows the same three activities, "V/S (constrained)," "V/S (*un*constrained)," and Capture and differs from the machine model only by the event labels on the two outgoing transitions from "Capture." The symbol "--**>**" denotes that the pilot sets the newly set altitude ahead of the current aircraft altitude and "**<--**" when he or she set the altitude behind the current aircraft altitude.

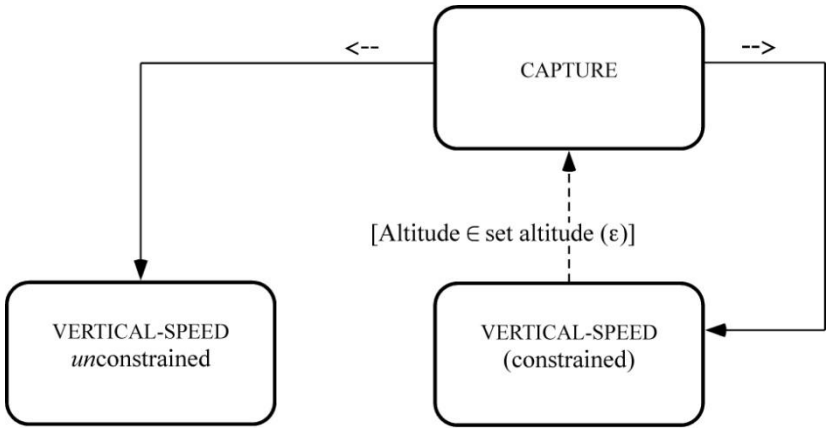


Figure 3(b). User model of vertical modes.

The user model differs from the machine-model in the way in that the "critical altitude" is evaluated. In the machine model this critical value is the aircraft altitude at the time of entry into the capture activity (capture start). In the user model this critical altitude is the current altitude of the aircraft at the time of changing the altitude setting (aircraft altitude). These two values are inconsistent when the new altitude setting is ahead of the capture start value and behind the current aircraft altitude (marked in gray in Figure 3(c)).

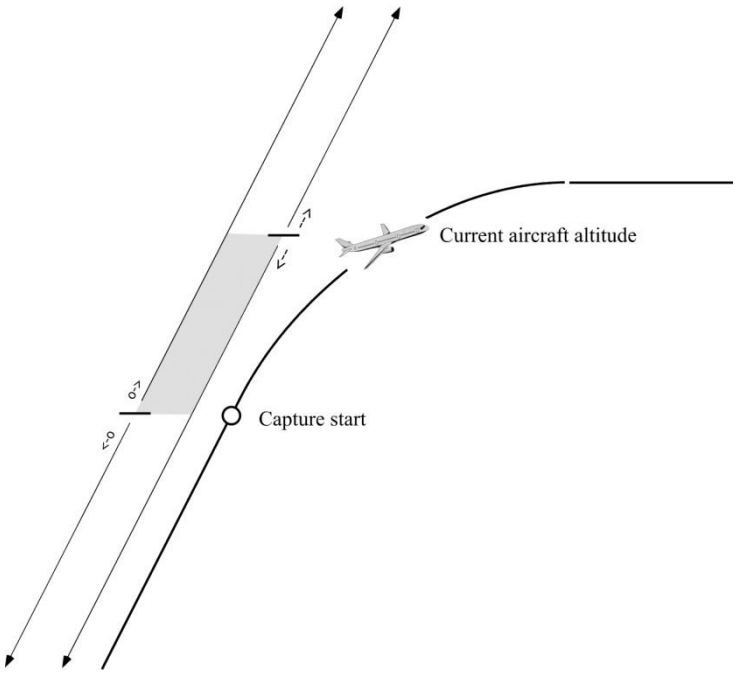


Figure 3(c). Capture profile (during climb).

Formal Verification

We turn now to a formal verification of the autopilot fragment discussed above. The task specification consists of the requirement that the pilot be able to predict whether, as a consequence of an altitude setting change, the target altitude is captured or not. The composite model is depicted in Figure 13(d) and shows explicitly how the error state occurs. The event labels in the composite model are presented as the user-model events (and the corresponding machine events are in parentheses.)

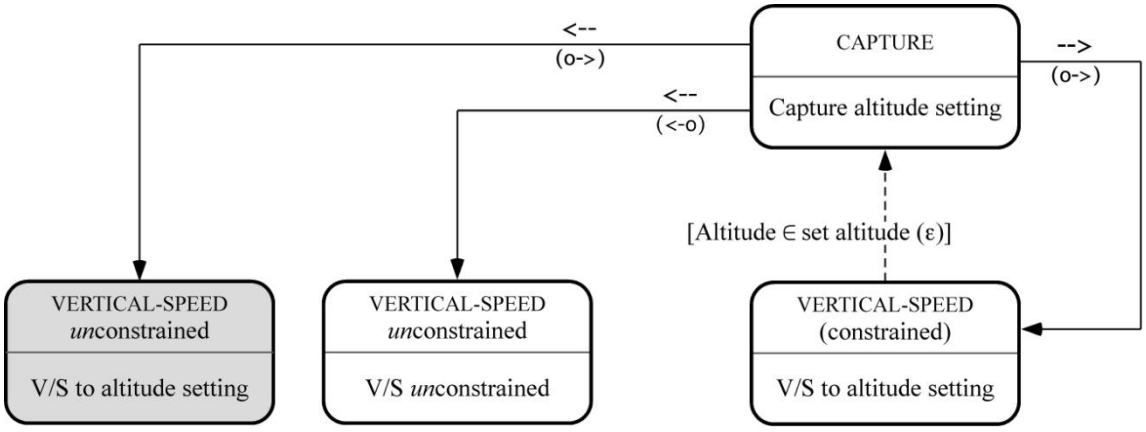


Figure 3(d). Composite model of vertical modes.

When the newly set altitude is ahead of the current aircraft altitude it is always also ahead of the capture start altitude. Similarly, when the new altitude is behind the capture start altitude it is also always behind the current aircraft altitude. However, when the newly set altitude is behind the current value, it can still be ahead of the capture start altitude. The result is a contradiction because the user model indicates unconstrained flight while the autopilot is set to capture (Degani et al., 2000).

Generation of User Models and Interfaces

In this Section we discuss a methodology for generation of user models. The objective of the methodology is to derive a user model that is correct for the specified tasks; namely, that it is free of error states and contains no problematic event discrepancies. A second requirement is that this user model should be succinct. That is, we want the model to be as simple as possible in terms of number of states, transitions and events necessary to describe and operate the system.

The methodology for satisfying these two requirements is based on a systematic method for reducing a detailed state transition (machine) model of the system into a smaller model and then, if possible, omitting or abstracting some of its transitions and events. The end result is a "most" succinct user model, which then serves as the basis for the interface construction.

As in the problem of verification, the methodology requires that the machine's behavior and user's tasks be fully specified. We assume, a priori, that the user's tasks are within the machine's capabilities, and that the machine's responses to user-triggered events be deterministic.

The conceptual approach for generating correct and succinct user-models is based on the observation that while the user-model must enable correct interaction with the machine, the user may not need to track every internal state of the machine. Thus, two internal states, A and B belonging to the same specification class, can be grouped together and represented as a single state if the intrinsic details of whether the current internal state is A or B are inconsequential to the user (in that the system's responses would be the same from the user's perspective).

Formally we say that two states A and B need not be distinguished, whenever (1) they both belong to the same specification class, (2) each *user triggered* event that is available and active in A is also available and active in B, and (3) whenever starting from either of the two states and triggering by the same event sequence, the state pairs visited, respectively, also satisfy conditions (1) and (2). State pairs that satisfy the above three conditions need not be distinguished by the user and are

referred to as "compatible." State pairs that do not satisfy the above three conditions are referred to as "incompatible".

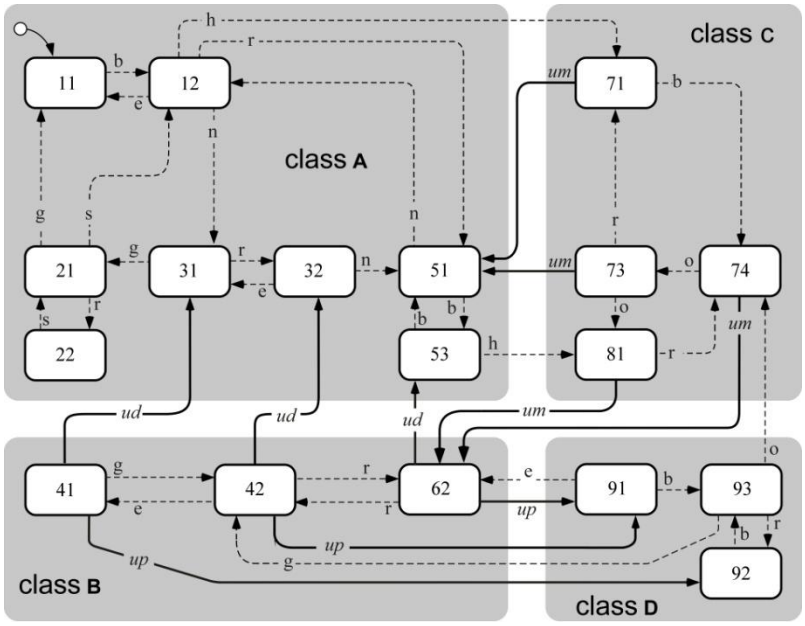
An efficient algorithm for computing such compatible and incompatible state-pairs is based on the use of merger tables (Kohavi, 1978; Paull, and Unger, 1959). A merger table is a table of cells that lists for each state pair of the machine, the set of all distinct state pairs that are reached through a single common transition event. For a machine with n states, there are n*[n-1]/2 cells in the table, and by iteratively stepping through the table one event transition at a time, all incompatible state pairs are detected, thereby "resolving" the table. All state-pairs that are not found to be incompatible are designated as compatible. From these compatible pairs, the largest possible sets of compatible states, i.e., triplets, quadruples, quintuples, etc., are created. These sets of compatible states are called "maximal compatible" sets and can be thought of as the building blocks of the user model (see Heymann & Degani, 2007 for the details of the algorithmic procedure for generating compatibles). The next step consists of collecting from the maximal compatibles a suitable subset adequate for construction of a correct and succinct user model. The formal criterion for such a selection is that the selected subset of maximal compatibles, constitute a "cover" of the original machine's state set. That is, each state of the original machine must be a member of at least one maximal compatible of the selected subset. To obtain a most succinct user model, we do not want just any cover, but rather a minimal one. That is, a cover such that none of the selected maximal compatibles can be omitted from the selected set without violating the cover property. A second condition that we need to impose is that the set of target states of each transition emanating from a maximal compatible be included in a maximal compatible of the selected cover. This condition may sometimes be inconsistent with the cover's minimality condition in which case additional maximal compatibles are incrementally added to the minimal cover. (In the worst case, this incremental addition of maximal compatibles will terminate successfully when all maximal compatibles are chosen).

Once the suitable set of maximal compatibles has been selected as just described, the next step is to determine the transitions in this reduced model. These are defined to be consistent with the original machine model and with the partition of the state set into specification classes. The resultant model constitutes the user model from which the interface is constructed.

Running Example

Figure 4 is a description of a machine model that contains 18 states, 42 transitions, and 11 event labels. Some of the transitions are manually triggered (*ud*, *um*, and *up*) while the rest are automatic.

Four separate specification classes are defined for this machine: A, B, C, and D as highlighted in gray in the Figure. The task requirement is that at each instant the user be able to predict the next specification class that the machine will enter either automatically or as a result of his or her



interactions.

Figure 4. Machine model and specification classes.

Upon applying the algorithmic procedure for computing compatible sets to the machine model described above, the following eight maximal compatibles are obtained (Figure 5(a)):

1. (11, 12, 21, 22, 31, 32);
2. (12, 21, 22, 31, 32, 51);
3. (11, 21, 22, 31, 32, 53);
4. (21, 22, 31, 32, 51, 53)
5. (41, 42, 62);
6. (71, 73);
7. (74, 81);
8. (91, 92, 93).

Fig. 5(a). Eight maximal compatibles

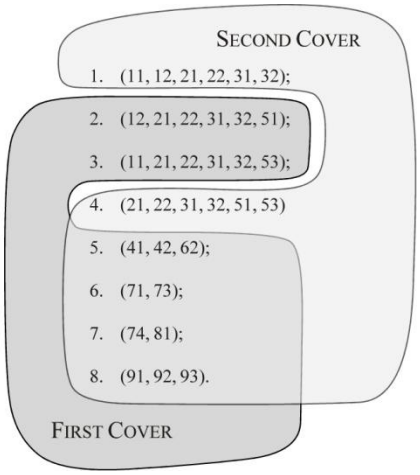


Fig. 5(b) Two minimal covers.

One interesting observation is that several internal states appear in more than one maximal compatible. For example, internal states 21, 22, 31 and 32 appear in maximal compatibles 1, 2, 3, and 4, and internal state 11 appears in maximal compatible 1 and 3. Also, there are many sets of maximal compatibles that contain all the internal states of the machine and hence constitute covers. Two of these covers, {1,4,5,6,7,8} and {2,3,5,6,7,8}, are minimal (Figure 5(b)).

The selection among the various minimal covers as candidates for user-model construction cannot be quantified. Various kinds of engineering and human factors design considerations can be applied towards this decision, such as the number and intuitive nature of the states and transitions in each candidate cover, the physical interpretation of the reduced model, and technical ease of implementation. Of course, when no profound reason exists to prefer one cover over another, any cover may be selected. In the current example we selected the cover that consists of the maximal compatibles {2,3,5,6,7,8} for reasons that will be explained later.

Constructing the Modes and Transitions in the User Model

We now proceed to construct the user model. We first designate the selected maximal compatibles (2,3,5,6,7,8 in our case) as user model states, sometimes referred to as modes. Thus, maximal compatibles 2 and 3 are designated as modes A-1 and A-2 indicating that they belong to specification class A. Maximal compatible 5 is mode B, and maximal compatibles 6 and 7 are designated as modes C-1 and C-2 respectively. The last maximal compatible, 8, is mode D.

MODE A-1	(71)	(51, 22, 32)	(31, 51, 12)	(11, 31)	(11, 21)	(12, 21)	(53)
2. (12, 21, 22, 31, 32, 51)	h	r	n	e	g	s	b
MODE A-2	(12, 51)	(11, 21)	(12, 21)	(22, 32)	(31)	(51)	(81)
3. (11, 21, 22, 31, 32, 53)	b	g	s	r	e	n	h
MODE B	(31, 32, 53)	(42)	(92, 91)	(62, 42)	(41)		
5. (41, 42, 62)	ud	g	up	r	e		
MODE C-1	(74)	(51)	(71)	(81)			
6. (71, 73)	b	um	r	o			
MODE C-2	(62)	(73)	(74)				
7. (74, 81)	um	o	r				
MODE D	(93)	(62)	(74)	(92)	(42)		
8. (91, 92, 93)	b	e	o	r	g		

Fig. 6. User model modes, their respective event labels, and resultant machine model target states.

The next step is establishing the transitions of the user model. We begin by building a table that lists all user model modes. For each mode, the events that emanate from its constituent states are

listed as well as their target state(s). The resulting table is presented in Figure 6, where for example, event *r* emanates from (the constituent states of) mode A-1 to states 51, 22, 32.

Next we create the transitions between the modes of the user model as follows: for each mode (e.g., A-1), each emanating event (*h,r,n,e,g,s,b,* from A-1) is drawn to a mode that includes *all* the target state(s) of that transition (as listed in the parenthesis above each event in Figure 6). For example, the transition labelled *r* goes from A-1 to A-1 as a self loop because A-1 includes the three target states (51, 22,32). The complete user model is depicted in Figure 7(a).

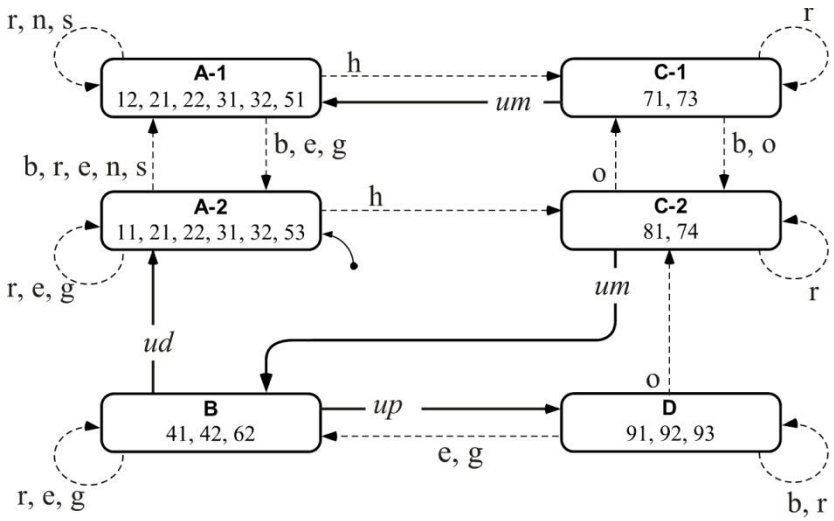


Figure 7(a). User model modes, transitions, and all event labels.

Event Abstraction

While it is possible to accept the model in Figure 7(a) as the user model (and as the foundation for the interface), there are several possibilities for further reduction and simplification of the model. We describe here three such possibilities: (1) elimination of non-determinism, (2) event label grouping, and (3) selected self-loop removal.

With respect to the first possibility -- elimination of non-determinism -- note in Figure 7(a) the existence of non-deterministic transitions between modes. For example, event label *r* emanates from mode A-2 to A-1 and also appears on the self-loop around A-2. Such non-determinism is the by-product of the reduction process wherein many internal states (with their respective event transitions) are encapsulated into a single mode. (This, in view of the principles of reduction process, is possible only within a given specification class).

Within the confines of specification class A, it does not really matter whether the system remains in mode A-2 or transitions to A-1 in response to transition *r*. We can thus eliminate this non-determinism by deleting one of the *r* labels--either the one transitioning to A-1 or the one on the self-loop. In general, we would prefer to delete transition between modes, and keep transitions that self-loop around a mode, because self-loop transitions do not produce mode switching on the interface. Thus, when the redundant event label *r* from A-2 to A-1 is deleted, *r* remains only in the self-loop. Using the same logic, we delete event label *e* of the transition from A-2 to A-1, leaving it only on the self-loop around A-2 (see Figure 7(b)).

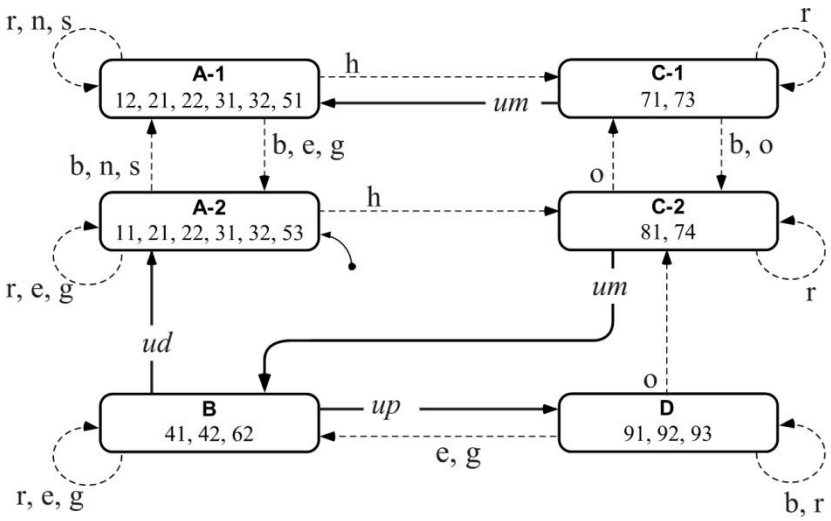


Figure 7(b). After eliminating of non-deterministic event labels e and r (from A-2 to A-1).

Event label grouping is the second possibility to further abstract the model. As can be seen in Fig. 7(b), event labels *e* and *g* always appear together. Nowhere in the model do we see *e* or *g* separately; whenever label *e* is enabled in a mode so is *g*. Thus, groups of event labels that always appear together on a set of transitions can be abstracted into a single representative label because the distinction between them is inconsequential to the user who is monitoring such mode switching. Therefore, event labels *e* and *g* (outgoing from mode A-1, A-2, B, and D) can be abstracted into an event that we (arbitrarily) label as *q*. Similarly, event labels *n* and *s* (on the outgoing transition from A-1 and A-2) are abstracted into *p*. The results of these event groupings appear in Figure 7(c).

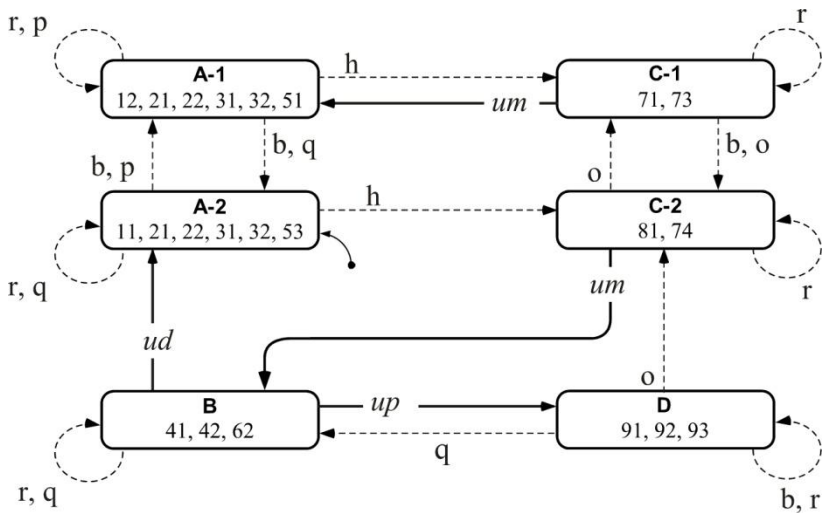


Fig. 7(c). Event label groupings (q and p).

Finally, the third possibility looks at events that occur on self-loops, which have no practical effect on the user model as no mode transition occurs as consequence. Hence in many cases (but not always, e.g., when timing events occur or when some mode related information gets updated) we have an opportunity to eliminate such selected self-loop events from the user model. As a consequence event labels that appear only in self-loops can be completely removed (e.g., event *r*). Event labels *p, q,* and *b* can be removed from self-loops (but one has to be careful not to remove them from inter-mode transitions). In the model presented in Fig. 7(d), all these self-loop events were indeed removed. The result is a correct and succinct user model that has been reduced to contain 6 modes, 12 transitions, and only 8 different event labels.

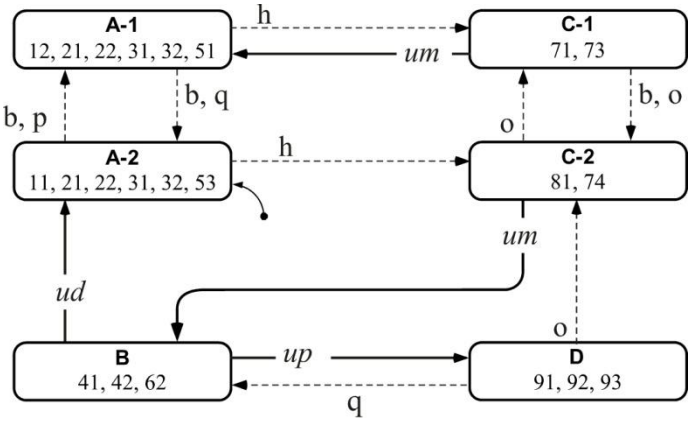


Fig. 7. The final user model.

The user model of Figure 7(d) has been obtained by starting from a set of maximal compatibles 2,3,5,6,7,8 that constituted a minimal cover of the state set (Figure 5(a)). Selecting a minimal cover

is always advisable because it will generally lead to a succinct user model. However, not every initial minimal cover is necessarily a sufficient set of maximal compatibles for the model construction. Sometimes a minimal cover may require augmentation with additional maximal compatibles. For example, had we started the model construction with the minimal cover consisting of maximal compatibles 1,4,5,6,7,8, we would have discovered that the target set (of states) reached from the constituent states of compatible 4 in response to event *n* consists of the states 51 and 12. This target set is not included in any of the maximal compatibles of the cover 1,4,5,6,7,8 but is included in maximal compatible 2. Therefore, maximal compatible 2 must be added. The resulting set (now consisting of maximal compatible 1,2,4,5,6,7,8) is no longer a *minimal cover*, but rather a *sufficient cover* for the model.

Related Work

David Parnas (1969) was probably the first to apply formal methods to user interaction with a computer. Using the finite state machine formalism, he illustrated several design errors such as "almost-alike" states, inconsistent ways to reach states, and user input problems. Foley and Wallace (1974) also used the finite state machine formalism to develop a language for human-computer interaction. Jacob (1983) used the same formalism for designing specifications for user interaction with a complex communication system (1986), later extending the approach to model direct manipulation of user interfaces (cf. Wasserman, 1985). During the same period, Mackinlay (1986) described a "composition algebra" to support the formalization of graphical user interface designs enabling a coherent flow from specification to implementation.

Reisner (1981, 1982) articulated the goal of designing better user interfaces by adapting formal methods from mainstream computer science. She discussed key technical challenges that are still relevant today: How can formal models be integrated with models of attention, perception, cognition, and memory? How can discrete models be integrated with models of time and other continuous variables? How can models be used during the design phase to make good decisions that could not be made using common sense or other less costly methods?

The study of formal models for human-interactive systems expanded rapidly during the 1990s with some 200 articles and books on this topic. Harrison and Thimbleby (1990) review early work on formal methods for the design and analysis of GUIs, procedures, and interactive systems (see also Harrison & Duke, 1995). Bredereke and Lankenau (2004) review more recent work on the analysis and correction of design errors in mode-based systems.

Hinze, Malik, and Malik (2006) apply formal methods to the design of a mobile, context-sensitive tourist information system (TIP) for implementation on small-screen devices. They conclude:

"The formal modeling approach has greatly improved the system design. By describing the requirements formally, developers are forced to think carefully about the planned system. The activity of specifying and modeling on its own helps to understand and clarify many aspects of the system to be constructed... Exhaustive simulation and analysis of the model allows for finding subtle problems at this very early stage of the design. Several issues with the proposed interaction protocols have been discovered and solved, which may otherwise have remained undetected until much later during the implementation of the system."

John Rushby and his colleagues (Crow, Javaux & Rushby, 2000; Rushby, 2001, 2002) analyzed human-automation interaction, demonstrating the use of theorem-provers and model-checkers to explain deviations of pilots' mental models from correct models of autopilot behavior. They showed how formal methods could be used in a cycle of analysis, re-design, and re-analysis to improve a human-machine system. Sherry, Medina, Feary, and Otiker (2008) address issues of scale and complexity in the next-generation air-traffic management system. They propose formal models that can help estimate time, cost, and risk parameters for system-level changes entailed by partially automated subsystems. These models can also help identify functions that are not supported by the automation.

Combéfis and Pecheur (2009) use a formal approach to user-modeling. They propose general properties of "good" mental models, as well as a method for generating mental models. This work extends earlier work in which informal or handcrafted formal mental models are used. Their methodology increases the objectivity of model-checking using composite user-system-interface behavioral models, while at the same time addressing problems such as how to model the actual observation of system-state by the user, in contrast to theoretical observability.

Rukšėnas, Curzon, Back, and Blandford (2008; cf. Su, Bowman, & Barnard, 2008) integrate a psychologically oriented architecture with a model-checking environment and apply this system to the analysis of human error. Separating the device model from the user model, they focus on the user's interpretation of device behaviors. By increasing the psychological realism of the user model, they identify potential user errors triggered by misinterpretation or confusion. They discuss a broader design and analysis framework, which integrates theorem-proving, model-checking, design rules, and psychological models at increasing levels of realism. "...[Even a] small number of principles is rich enough for plausible erroneous behavior to emerge that was not directly expected."

Drewes (2006) and others have recently generalized the approach of describing user interfaces with formal grammars. This work aims to solve some of the problems of visualizing multivariate time series, ontologies, and other complex data structures. The research maintains definite continuity with the early work of Reisner, Mackinlay, Larkin and Simon (1987), Casner (1991), and others, while aiming at higher levels of complexity in terms of both system size and hybrid structure.

Most of the work discussed above has used some variant of concurrent, communicating finite-state machines, direct descendants of Petri nets. The models are framed in terms of discrete mathematics, and they do not explicitly represent continuous variables, such as time, probability, or graded response potentials (Jamieson & Vicente, 2005). Tomlin, Mitchell, Bayen, and Oishi (2003) provide an overview of current methods for modeling and verifying hybrid systems, using a commercial jet autopilot as a case study. Oishi, Mitchell, Tomlin, & Saint-Pierre (2006) show how to prove simultaneous satisfaction of envelope protection and stability requirements, illustrating their method with a two-aircraft collision avoidance scenario.

Today there is no doubt that formal methods can provide insights into complex human-interactive systems -- insights unavailable to intuitive or even quantitative analyses (Degani, 2004; Degani, Heymann & Gellatly, 2011; Leveson, 2009). On the other hand, some of the early issues raised by Reisner (1981, 1982) still remain with respect to hybrid systems, psychological factors, user expectation and population stereotypes, scaling up to realistic systems (Berstel, Reghizzi, Roussel & San Pietro, 2005), and making formal methods accessible to and usable by non-specialists (Chakrabarti & Sukumaran, 2009; Gow & Thimbleby, 2006; Gow, Thimbleby, & Cairns, 2006).

Conclusions and Future Work

In this chapter we introduced a formal methodology for analysis and design of human-automation interaction. Specifically we focused our attention on the issue of whether a given interface of a complex system such as an aircraft autopilot can be operated correctly and reliably under all operating conditions without the possible occurrence of unexpected automation responses. We also dealt with the problem of generating efficient (succinct) interfaces for such systems starting with a detailed knowledge of the system's behavior.

Key to the verification methodology are two models. The first is a model of the machine, which describes all the relevant behavior and responses of the system and is obtained from engineering specifications. The second is a user model which is a formal description of all the information provided to the user by the manufacturer regarding the operation of the system under consideration.

Such information is frequently given in user manuals, operating instructions, training, and may also involve user expectations. The verification is based on a concurrent execution of the two models viewed from the user's perspective. This verification procedure is aimed at detecting interface discrepancies such as "error states" and "event discrepancies." The methodology is described in detail for systems that are modeled as simple finite state machines. The behavioral specifications are given in terms of a partition of the state space into distinct classes that need to be distinguished by the user.

The generation of interfaces is based on a systematic top-down reduction procedure of a detailed model of the machine under consideration. The customary interface design approach is a bottom-up process in which engineering knowledge of the system and a trial-and-error evaluation is used to generate the interface. The top-down reduction procedure presented herein guarantees that the resultant user model, which serves as the basis for the interface design process, is both correct and succinct. The procedure is again demonstrated by modeling the systems as a simple finite state machine and the specification in terms of distinct classes. While we did not address here the potential problem of computational complexity of reducing large state spaces, a computerized tool developed to generate interfaces was successfully tested on systems containing several hundred states (Shiffman, Degani, & Heymann, 2005).

Our emphasis in this chapter has been on *formal methods* based on the premise that both the system and the operational requirements can be modeled in a rigorous mathematical framework. While such an approach has been common in engineering practice and computer science, the research described in this chapter involves extending these methods to human-interactive systems, thus broadening the definition of system under study to include human operators. Various formalisms are available for representing the state-space. While we have employed the finite state machine formalism to demonstrate our approach, the basic verification and interface design approach can be readily generalized and extended to more complex system models and specifications.

The methodology for modeling and analysis of human machine interaction described in the present chapter constitutes a fruitful foundation for design and analysis of a wide range of applications where humans interact with machines. In addition to aviation, a promising domain of application is the automotive industry where increasingly sophisticated automation aids (e.g., lane keeping, lane changing, and adaptive speed control systems) are being tested and introduced. New computational and informational devices that are being introduced in the automotive industry (collectively called *infotainment systems*) are also amenable to the kind of user interaction correctness analysis

described above. Similar information systems are introduced in aviation (e.g., electronic flight bags) and in the medical field.

Below are a few chosen extensions of the work described in this chapter:

1. Expansion of the verification methodology for exploiting the existence of potentially problematic behaviors of user models. Focus should be placed on ambiguous behaviors related to unexpected interactions from the point of view of the user.
2. Development of a formal tool for automatic verification of user machine interfaces in the spirit of the methodology described and explained in this chapter.
3. Development of a practical tool for implementation of system reduction and interface design. The methodology described here has been tested and applied manually but for larger system an automatic tool is desirable.
4. Development of a formal approach for the creation and categorization of design patterns in the context of user interaction, as well as a modeling tool for implementation and verification of design patterns and identification of opportunities for their implentation.
5. Development of additional user-interaction correctness criteria and possibly extending the set of criterions to include "softer" properties such as population stereotypes and user expectations, as well as some basic rules of what is an "acceptable" as well as "desirable" user interaction design.

As a final comment, the approach and methodology presented here can serve as a starting point in the development of future adaptive automation wherein system and interface reconfiguration will take place in response to evolving operating conditions as well as users' preferences, needs, and information requirements.

References

Bayen, A.M., Mitchell, I.M., Oishi, M.M.K. & Tomlin, C.J. (2007). Aircraft autolander safety analysis through optimal control-based reach set computation. *Journal of Guidance, Control, and Dynamics*, **30**(1), 68-77.

Berstel, J., Reghizzi, S.C., Roussel, G., San Pietro, P. (2005). A scalable formal method for design and automatic checking of user interfaces. *ACM Transactions on Software Engineering and Methodology*, **14**(2), 124–167.

Bredereke, J., & Lankenau, A. (2004). Safety-relevant mode confusions: Modelling and reducing them. *Reliability Engineering & System Safety*, **88**(3), 229-245.

Casner, S.M. (1991). Task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, **10**(2), 111-151.

Chakrabarti, S.K., & Sukumaran, S. (2009). Using spreadsheets for finite state modelling. *Proceedings of the 2nd Annual Conference on India Software Engineering*. New York: ACM, 27-36.

Combéfis, S., & Pecheur, C. (2009). A bisimulation-based approach to the analysis of human-computer interaction. In G. Calvary, T.C.N. Graham, & P. Gray (Eds.), *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York: ACM, 101-110.

Crow, J., Javaux, D., & Rushby, J. (2000). Models and mechanized methods that integrate human factors into automation design. In K. Abbott, J.J. Speyer, & G. Boy (Eds.), *Proceedings of HCI-AERO'2000*. Toulouse, France: Cépaduès Éditions, 163-168.

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

Degani, A., Heymann, M., Meyer, G., & Shafto, M. (2000). *Some formal aspects of human-automation interaction. NASA Technical Memorandum #209600*. Moffett Field, CA: NASA Ames Research Center.

Degani, A., Heymann, M., & Gellatly, A. (accepted). HMI Aspects of Automotive Climate Control Systems. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. October 9-12, Anchorage, AL.

Degani, A. (2004). *Taming HAL: Designing Interfaces Beyond 2001*. New York: St. Martin's Press/Palgrave-Macmillan.

Drewes, F. (2006). *Grammatical Picture Generation: A Tree-Based Approach*. Berlin: Springer.

*** Endsley, Wickens, Salvucci, this *Handbook*

Gow, J., Thimbleby, H., & Cairns, P. (2006). Automatic critiques of interface modes. In S.W. Gilroy & M.D. Harrison (Eds.), *Interactive Systems*. Berlin: Springer, 201-212.

Gow, J., & Thimbleby, H. (2006). MAUI: An interface design tool based on matrix algebra. In R. Jacob, Q. Limbourg & J. Vanderdonckt (Eds.), *Computer-Aided Design of User Interfaces IV*. Kluwer, 81-94.

Harrison, M., & Thimbleby, H. (1990). *Formal Methods in Human Computer Interaction*. Cambridge, UK: Cambridge University Press.

Harrison, M.D., & Duke, D.J. (1995). A review of formalisms for describing interactive behaviour. In R.N. Taylor & J. Coutaz (Eds.), *Software Engineering and Human-Computer Interaction*. Berlin: Springer, 49-75.

Heymann, M., & Degani, A. (2007). Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm. *Human Factors*, **49**, 311-330.

Hinze, A., Malik, P., & Malik, R. (2006). Interaction design for a mobile context-aware system using discrete event modelling. In V. Estivill-Castro & G. Dobbie (Eds.), *Proceedings of the 29th Australasian Computer Science Conference*, **48**, 257-266.

Jacob, R. J. K. (1983). Using formal specifications in the design of human-computer interfaces. *Communications of the ACM*, **26**(4), 259-264.

Jacob, R.J.K. (1986). A specification language for direct-manipulation user interfaces. *ACM Transactions on Graphics*, **5**(4), 283-317.

Jamieson, G.A., & Vicente, K.J. (2005). Designing effective human-automation-plant interfaces: A control-theoretic perspective. *Human Factors*, **47**(1), 12-34.

Kohavi, Z. (1978). *Switching and Finite Automata Theory*. New York: McGraw-Hill.

Larkin, J.H., & Simon, H.A. (1987). Why a diagram is (sometimes) worth 10,000 words. *Cognitive Science*, **11**, 65-100.

Leveson, N.G. (2009). The need for new paradigms in safety engineering. In C. Dale & T. Anderson (Eds.), *Safety-Critical Systems: Problems, Process and Practice*. London: Springer, 3-20.

Mackinlay, J.D. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, **5**(2), 110-141.

National Transportation Safety Board. Aircraft incident report: Aeromexico DC-10-30, XA-DUH over Luxembourg, Europe, November 11, 1979. (NTSB-AAR-80-10). Washington, D.C.: transportation safety board, November, 1980.

Oishi, M., Mitchell, I., Tomlin, C., & Saint-Pierre, P. (2006). Computing viable sets and reachable sets to design feedback linearizing control laws under saturation. *Proceedings of the 45th IEEE Conference on Decision & Control*, 3801-3807.

Parnas, D. (1969). On the use of transition diagrams in the design of a user interface for an interactive computer system. *Proceedings of the 24th Annual ACM Conference*, 379-385.

Paull, M.C., & Unger, S.H. (1959). Minimizing the number of states in incompletely specified sequential switching functions. *Institute of Radio Engineers Transactions on Electronic Computers*, 356-367.

Reisner, P. (1981). Formal grammar and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering*, **7**(2), 229-240.

Reisner, P. (1982). Further developments toward using formal grammar as a design tool. *Proceedings of the 1982 Conference on Human Factors in Computing Systems*. New York: ACM, 304-308.

*** Ritter et al., this *Handbook*

Rukšėnas, R, Back, J., Curzon, P., & Blandford, A. (2008). Formal modelling of salience and cognitive load. *Electronic Notes in Theoretical Computer Science*, **208**, 57–75.

Rushby, J. (2001). Analyzing cockpit interfaces using formal methods. *Electronic Notes in Theoretical Computer Science*, **43**. URL: http://www.elsevier.nl/locate/entcs/volume43.html doi:10.1016/S1571-0661(04)80891-0

Rushby, J. (2002). Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, **75**(2), 167–177.

Sherry, L., Medina, M., Feary, M., & Otiker, J. (2008). Automated tool for task analysis of NextGen automation. *Proceedings of the 2008 Integrated Communications, Navigation and Surveillance Conference (ICNS 2008)*, 1-9.

Shiffman, S., Degani, A., & Heymann, M. (2005). UIVerify - a web-based tool for verification and automatic generation of user interfaces. *Proceedings of the 8th Annual Applied Ergonomics Conference*. New Orleans, LA.

Su, L., Bowman, H., & Barnard, P.J. (2008). Performance of reactive interfaces in stimulus rich environments: Applying formal methods and cognitive frameworks. *Electronic Notes in Theoretical Computer Science*, **208**, 95–111.

Tomlin, C.J., Mitchell, I., Bayen, A.M., & Oishi, M. (2003). Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, **91**(7), 986-1001.

Vicente, K.J. (2006). Cognitive engineering: A theoretical framework and three case studies. *International Journal of Industrial and Systems Engineering*, **1**(1-2), 168-181.