Formal Analysis and Automatic Generation of User Interfaces: Approach, Methodology, and an Algorithm

Michael Heymann, Technion, Israel Institute of Technology, Haifa, Israel, and **Asaf Degani,** NASA Ames Research Center, Mountain View, California

Objective: We present a formal approach and methodology for the analysis and generation of user interfaces, with special emphasis on human-automation interaction. **Background:** A conceptual approach for modeling, analyzing, and verifying the information content of user interfaces is discussed. **Methods:** The proposed methodology is based on two criteria: First, the interface must be correct – that is, given the interface indications and all related information (user manuals, training material, etc.), the user must be able to successfully perform the specified tasks. Second, the interface and related information must be succinct – that is, the amount of information (mode indications, mode buttons, parameter settings, etc.) presented to the user must be reduced (abstracted) to the minimum necessary. **Results:** A step-by-step procedure for generating the information content of the interface that is both correct and succinct is presented and then explained and illustrated via two examples. **Conclusions:** Every user interface is an abstract description of the underlying system. The correspondence between the abstracted information presented to the user and the underlying behavior of a given machine can be analyzed and addressed formally. **Applications:** The procedure for generating the information content of user interfaces can be automated, and a software tool for its implementation has been developed. Potential application areas include adaptive interface systems and customized/personalized interfaces.

INTRODUCTION

Human interaction with computers is so widespread that almost every aspect of our lives involves interaction with devices, information systems, and automated control systems. These computer-based machines have complex behaviors that comprise numerous internal states and events. Yet, the only “face” the user sees is the interface, always a highly abstracted description of the underlying machine behavior. This abstraction is inevitable because otherwise the user would be subjected to an enormous, and mostly irrelevant, amount of information. As such, an important and fundamental aspect of interface design involves an intricate process of abstracting information so as to suppress irrelevant information and retain the important information. The end result of this process is the information provided to the user on the interface. We argue that every interface designer, explicitly or implicitly, goes through this process of abstraction in his or her attempt to make user interaction efficient, reliable, and safe.

From this perspective, the designer’s goal is to strike a fine balance between providing too much information (some of which may be unnecessary to operate the machine) and providing insufficient information (thereby preventing the user from operating the machine correctly). Specifically, when insufficient information is provided to the user, he or she may not be able to perform the specified task correctly (e.g., determine the current mode of the machine and anticipate its next mode as a consequence of user interaction). As a result, either the user will be unable to perform the desired task altogether or there will be unexpected, faulty, and potentially dangerous outcomes. To illustrate this issue of correctness, consider the following example:

A modern 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

Address correspondence to Asaf Degani, Computational Sciences Division, NASA Ames Research Center, Mountain View, CA 94035-1000; adegani@mail.arc.nasa.gov. ***HUMAN FACTORS,*** Vol. 49, No. 2, April 2007, pp. 311–330. Copyright © 2007, Human Factors and Ergonomics Society. All rights reserved.

feet. The pilot enters the 10,000-foot altitude constraint into the autopilot, engages a mode called "Vertical Speed," and then selects the rate of climb (e.g., 2,000 feet/min); now the aircraft begins climbing to 10,000. 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.

Under one set of conditions, the aircraft will continue climbing (at the selected rate of climb of 2,000 feet/min) indefinitely; the aircraft will climb past 10,000 feet, and unless some control action is taken by the crew, the aircraft will keep on climbing. In another set of conditions, given the same pilot input, the aircraft will descend (from 9,000 feet) and then level off at 8,000 feet. Hence the same pilot input (entering the new 8,000 feet altitude constraint into the autopilot) triggers two different outcomes. Given the autopilot mode indications and displays and all related user manual information, it is impossible for the crew to determine what the aircraft will do.

The problem is not that the autopilot behaves in unpredictable and unexpected ways. The autopilot, in fact, is fully deterministic: If the newly entered altitude is above a certain reference altitude (the altitude at which the autopilot begins a gradual maneuver to capture and hold the target altitude), then the aircraft will descend and level off at 8,000 feet. However, if the newly entered altitude is below this reference altitude, the aircraft will continue climbing indefinitely. The problem is that this reference altitude value (which changes as a function of the aircraft's speed and altitude) is not available anywhere on the cockpit displays. Such an interface is formally defined as "incorrect" because the pilots, in the process of performing a specified task (climbing and leveling off), cannot anticipate the consequences of their interaction with the machine (Degani & Heymann, 2002).

In most practical systems, user interfaces do not provide a full and complete description of the underlying behavior of the machine with all its internal states, events, and parameters. Therefore, a major concern for designers is to make sure that the abstracted interface is indeed correct. Currently, this abstraction is performed in a heuristic and intuition-based manner. Its evaluation process usually involves many interface design iterations, costly simulations, and extensive testing. In industries such as medical equipment, nuclear systems, and commercial aviation, lengthy and complicated certification processes are in place to ensure that the system under consideration, its interface, and all user interaction aspects are safe and free of design errors. Yet, despite best efforts by design teams and certification officials, numerous incidents and accidents involving incorrect interfaces have been noted in avionics systems (Rodriguez et al., 2000; Rushby, 1999), maritime navigation systems (National Transportation Safety Board, 1997) and computer-based medical equipment (Leveson, 1995, Appendix A [the Therac-25 radiation machine]). Incorrect interfaces can also be found in Internet applications, automotive systems, and many consumer electronics devices (see Degani, 2004, for more than 10 examples).

On the flip side of the abstraction problem lies the case in which the interface provides too much information and overloads the user with superfluous and irrelevant information. Naturally, designers strive for interfaces (and user-manuals) that are not only correct but succinct. In most cases, a small set of modes is preferable to a large set of modes needed to perform a given task (Norman, 1983). Likewise, short sequences of user inputs are preferred over lengthy ones. The point here is not about eliminating functionality and user comprehension of the behavior of the system but, rather, about suppressing superfluous and irrelevant information that does not add much to the user's ability to control and manage the system (Thimbleby, Blandford, Cairns, Curzon, & Jones, 2002). The advantage of having succinct displays and shorter sequences of user inputs is that it minimizes the actual size of the user interface and the amount of indications that need to be designed and implemented. It also reduces the perceptual and cognitive burden on the user.

A Formal Approach to Human-Computer Interaction and a Literature Review

Many aspects of human-machine interaction, such as the design of interfaces in terms of their graphical appearance and layout, are empirical and, to some extent, artistic (Norman, 2004). Nevertheless, the information content of the interface can be described and analyzed using mathematical, or formal, methods.

Formal methods is a discipline for studying how mathematical models of systems can be used to develop efficient, reliable, and safe designs. Formal methods are employed to express design specifications and requirements as well as to perform systematic analysis and verification. Probably the earliest work in using formal methods to address human-computer interaction (HCI) issues was conducted by Parnas (1969), who used a finite state machine model to describe user interaction with a computer terminal. Using this modeling formalism, he was able to illustrate several design flaws such as "almost-alike" modes and inconsistent ways to reach a given mode. Foley and Wallace (1974) and Jacob (1983) used similar modeling formalisms for developing general interface design specifications for HCI. Jacob (1986) and Wasserman (1985) used formal methods for specifying direct manipulation aspects of user interaction in order to address the concurrent structure of multiple display objects (e.g., windows) that are open simultaneously.

By the mid-1980s, researchers in HCI began using formal methods as a way to analyze and measure user interaction. Kieras and Polson (1985) used formal methods to quantify the complexity of HCI. To do this, they modeled the device and the user's tasks as a finite state machine. Because both the device and the task were represented in the same formalism, they were able to identify cases in which the user's task structure did not correspond with the device's structure. Bösser and Melchoir (1992) employed the same approach and then applied graphing techniques to evaluate whether all the specified user's tasks could be achieved (given the device's functionality). Degani (1996) used a variant of a state transition system, called Statecharts (see Harel, 1987), to develop a framework that describes the environment, user's tasks, device functionality, and interface information as four concurrent processes; the intent was to understand automation-induced mode errors and to identify a variety of general interface ambiguity problems. Duke, Fields, and Harrison (1999) described a framework for modeling interactive computer systems in order to express HCI design specifications such as access control and information availability.

An important facet of formal methods is to prove that a given model of the system fulfills certain design criteria, or properties. In this context, a "property" can be a simple statement about something that the system model does (or does not) do. Extensive checking is then used to verify that the model of the system, for example, does not "deadlock" (see Dix, 1991; Harrison & Thimbleby, 1990; Palanque & Paternò, 1998; Paternò & Santoro, 2002). Rushby (1999, 2001) employed model checking techniques in order to detect inconsistencies between machine and user models by simultaneously tracking the operation of both models and then using an iterative search in order to modify the machine and user model so as to achieve consistency. Doherty, Campos, and Harrison (2000) used logical theorem-proving techniques to investigate the relation between system state behavior and user interfaces. Thimbleby et al. (2002) showed how unnecessary interface complexity imposed on the user may be inappropriate to the user's task needs and, more importantly, how an interface designed to hide irrelevant complexity had a beneficial impact on the overall reliability of the system.

Several research groups explored the use of algorithm-based processes for selecting and rendering display widgets. Szekely, Sukaviriya, Castells, Muthukumarasamy, and Slacher (1996) developed a framework (called MASTERMIND) for specifying the user's task, the functions of the system, and the requirement and style of the interface so as to create a model-based environment for user interface development. Browne, Davila, Rugaber, and Stirewalt (1997) used the MASTERMIND framework to develop an approach for automatically rendering user interfaces (e.g., dialogue boxes and file structure), given the underlying computer application. Bauer (1996) showed how a formal description of a computer application allowed for an automatic generation of interface widgets (mostly for dialogues and user input sequences). Krzystrof and Weld (2004) used an optimization algorithm for automatically selecting, resizing, and rendering display widgets to accommodate different display sizes (small cell phones, personal digital assistants, large computer screens, etc.).

Beyond formal interface descriptions, specification, evaluations, and algorithm-based rendering techniques, many other considerations must be taken into account to ensure efficient and successful human-machine interaction. These include cognitive and perceptual limitations, human physical abilities, redundancy of critical information, consistency, commonality with similar devices, training implications, and more. Nevertheless, at the foundation of any interface design rests the abstraction issue on which we focus our attention in this paper.

A FORMAL APPROACH FOR DESCRIBING HUMAN-AUTOMATION INTERACTION

The correspondence between the machine's behavior and the abstracted information that is provided to the user can be formally described and analyzed by considering the following four elements: the machine, the user's tasks, the user interface, and the user's model of the machine.

Machine

We consider machines that interact with their human users, the environment, and can act automatically. A widely used formalism to model machines is to describe them as state transition systems. A *state* represents a certain internal configuration of the machine. *Transitions* represent discrete state changes that occur in response to events that fire, or trigger, them. Some of these transitions occur only if triggered by the user, whereas others are triggered automatically. In general, we consider two types of automatic transitions: those that are triggered by the machine's internal dynamics (e.g., timed transitions) and those that are triggered by the external environment (e.g., the way an air-conditioning compressor is activated when the temperature reaches a set value).

To illustrate a typical machine model, consider Figure 1, which describes the behavior of a semi-automatic transmission system of a large vehicle. We shall use the convention in which *user-triggered* transitions are depicted as solid lines and *automatically triggered* transitions are depicted by broken lines. The transition lines are directed and are labeled by the triggering event that causes the machine to move from state to state.

The transmission system in Figure 1 has eight states. These states are grouped into three clusters, which we refer to as *modes:* Low, Medium, and High. Thus there are several internal "speed-level" states contained within each mode: low-1, low-2, low-3 in the Low mode; medium-1 and medium-2 in the Medium mode; and high-1, high-2, high-3 in the High mode. The system shifts automatically between these internal states (based on torque, throttle, engine rpm, and actual car speed). Automatic upshifts (to higher speed states) are denoted by the event *up,* and automatic downshifts (to lower speed states) are denoted by the event *down.*

The user interacts with the system by means of a gear lever: Pushing the lever up shifts to a higher torque level, and pulling it down shifts to a lower one (see Figure 1). These user-triggered transitions are denoted by events *push-up* and *pull-down,* respectively.

User's Tasks

Generally speaking, users interact with a machine to achieve a specific set of tasks (Parasuraman, Sheridan, & Wickens, 2000). These tasks vary widely, ranging from common tasks, such as

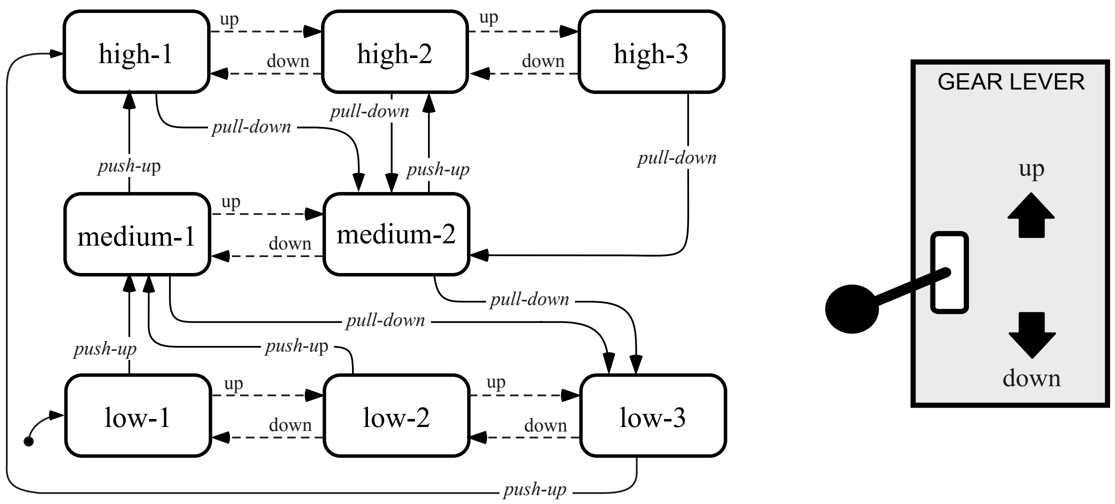

Figure 1. Transmission system of a vehicle (and the driver's gear lever).

using consumer electronic devices (e.g., VCRs) and interacting with Web browsers, to more complex tasks, such as operating safety-critical systems (e.g., medical devices and navigation systems aboard ships and aircraft). With respect to controlling and supervising automated systems, typical tasks involve monitoring a machine's mode changes (e.g., an automatic landing of an aircraft), execution of specific sequences of actions (e.g., making an online transaction), and supervising a system such that it does not enter into an illegal state (e.g., in process control).

It is possible to describe these tasks formally. This is done by first partitioning the entire machine's state-space into disjoint clusters that we call *specification classes.* A specification class is a set of internal states that the design team determined that the user need not distinguish among. For example, in the transmission system the three modes – Low, Medium, and High – are specification classes. (These are typically defined by design teams by using task analytical techniques and inputs from expert users.) Next, the design team specifies the task requirements. For example, one task requirement, which is common to almost all automated systems that are supervised by humans, is for the user to track these specification classes, unambiguously. In the case of the transmission system, the design team stated that the user must be able to determine whether the system is in, or is about to enter into, the Low, Medium, or High specification class. What this means is that the user is not required to track every internal state change of the machine (e.g., transitions between the states high-1, high-2, and high-3 which are contained in the High mode need not be tracked). Using this approach, one can also formally express other types of user's tasks, such as reliably executing a specified sequence of actions (Degani, Heymann, & Shafto, 1999).

Interface

In almost every machine, the events that take place inside the mechanism are purposefully abstracted and the interface displays only a limited view of these internal states. In most computer systems, a dedicated software collects events from the underlying machine and then passes this information to a special component that generates the display. In automated control systems such as autopilots and flight management systems, a display generator, located between the system and the interface, takes in selected events from the machine and provides outputs in the form of commands to light up (or turn off) display indications. As such, one aspect of the work described in this paper is to determine which events must be collected from the machine and then presented, in the form of indications, on the user interface.

To illustrate this formal approach for considering interfaces, we return to the vehicle transmission example. Figure 2a is a suggestion for a simple and straightforward user interface for the vehicle transmission system. Note that in this proposed design all internal transitions are removed from the interface and, consequently, from the user's awareness. As one can see by comparing Figure 2a with the machine model in Figure 1, the Low mode actually has three internal states: low-1, low-2, and low-3. When the user first enters manually into Low mode, low-1 is the active state (see the small quarter-circle arrow at the bottom left of Figure 1); when the driver increases speed, an automatic transition to low-2 takes place, yet this internal transition is not evident to the driver, who is only aware of being in the Low mode. The same applies to all other internal transitions in the system.

User Model

Manufacturers normally provide users with information about the working of the machine

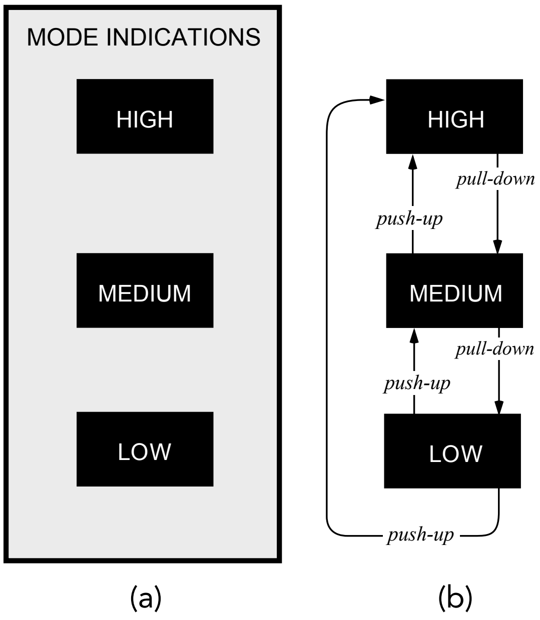

Figure 2. (a) Proposed interface. (b) The corresponding user model.

by means of user manuals, which describe the functions of the machine and its behavior as a consequence of user action and environmental conditions. Most verbal statements for consumer electronics, as well as for more complex systems (e.g., avionics), take the following form: "When the machine is in mode A and button X is pushed, the machines transitions to Mode B."

The user manual for the transmission system should be consistent with the interface of Figure 2a. It might tell the driver that when the transmission is in Medium mode, pushing the lever up would cause the system to shift to High mode, a downshift would transition the system to Low mode, and so on. These series of fragmented statements describe to the user how the machine works as well as how he or she is expected to interact with it. (Again, however, note that these user manual statements are abstractions of the actual behavior of the machine.)

In Figure 2b we incorporated all the user-triggered transitions of the machine with the three mode indications (low, medium, high). The resultant description shows how the user, when monitoring the machine through the proposed interface, would see the machine's behavior. We refer to this description of the interface indications, and of the transitions and events that drive them, as the *user model* of the machine.

The user model is based on the interface because it directly relates to the indications displayed there. Thus, as mentioned earlier and can be readily seen in Figure 2b, the interface is actually embedded in the user model. Therefore, for practical purposes, we will consider from here on only the user model in the process of analyzing and generating interfaces.

INTERFACE CORRECTNESS CRITERIA

For the purpose of the analysis, the machine model and user's tasks must be fully specified. (Our only assumption is that the machine's behavior is deterministic and the user's tasks are within the machine's abilities.) This leaves the user model (and the interface that is embedded in it) as the focus of the analysis.

One immediate observation about interface correctness is that the machine's response to user-triggered events must be deterministic. That is, there must not be a situation wherein, starting from the same mode, an identical user event (e.g., upshifting the gear) will sometimes transition the system into one mode (e.g., Medium) and at other times transition it into another (e.g., High).

Broadly speaking, there are three user-interface correctness criteria that should be satisfied in the process of analyzing and generating interfaces: An interface is correct if there are no *error states,* no *restricting states,* and no *augmenting states:*

- An error state occurs when the user interface indicates that the machine is in one mode when, in fact, the machine is in another. Interfaces with error states lead to faulty interaction. Frequently (but not always), error states are caused by the presence of nondeterministic responses to user interaction.
- A restricting state occurs when the user can trigger certain mode changes in the machine that are not present in the user model and interface. Interfaces with restricting states tend to surprise and confuse users.
- An augmenting state occurs when the user is told that certain transitions are available when, in fact, they cannot be executed by the machine (or are disabled). Interfaces with augmenting states puzzle users and have contributed to operational errors.

All three criteria can be expressed mathematically and therefore can be dealt with using formal methods of analysis (see Heymann & Degani, 2002).

Nondeterministic Interfaces and Error States

We begin by analyzing the proposed user model of Figure 2b. The manual upshift from Medium to High and the downshift from High to Medium and Medium to Low are always predictable; the user will be able to anticipate the next mode of the machine. However, note that the transitions out of Low depend on the internal states: upshifts from low-1 and low-2 switch the transmission to Medium, whereas the upshift from low-3 switches it to High (see Figure 1). Here, the same user-triggered event (*push-up*) takes the user to either of two different machine modes. However, because the display abstracts from the user which internal state the system is in, whether the system will transition to Medium or to High is unpredictable. In other words, the proposed display, with respect to user-triggered events, becomes nondeterministic and may lead to an error state. Therefore, one must conclude that the proposed interface and corresponding user model of Figure 2 is incorrect.

Next consider the alternative user model in Figure 3, in which the display has been modified

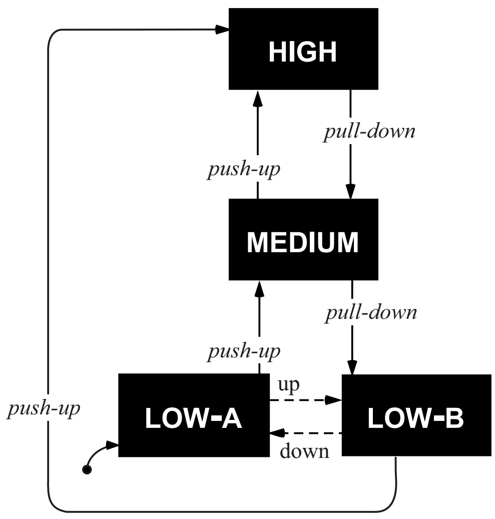

Figure 3. Alternative user model.

to partition the Low mode into two submodes (Low-A and Low-B). The user manual is modified correspondingly to explain to the user that the upshift (*push-up*) from Low-A transitions the system to Medium, whereas the upshift (*push-up*) from Low-B transitions the system to High. We will now try to analyze the correctness of this user model, but this time we proceed in a formal way (Degani & Heymann, 2002).

In any human-machine system, two concurrent processes are constantly at play: (a) the machine with its internal states and transitions and (b) the interface annunciations with the associated user model transitions. These two processes, or models, must "march" in synchronization and never encounter error states, restricting states, or augmenting states. Verification that this is indeed true can be accomplished by constructing a composite model that incorporates both the machine model states and the user model states. In this composition, we combine corresponding user model states and machine model states into state pairs and evaluate their synchronized march with respect to the specification classes and the task requirements.

The machine (see Figure 1) starts in state low-1, and the display and user model (see Figure 3) starts in Low-A, so the first composite state in Figure 4 is "low-1, Low-A." Upon an internally triggered automatic shift (event *up*), the machine transitions to low-2 and the display to Low-B. Now it is in composite state "low-2, Low-B," and all is well. Another internally triggered automatic transition *up* takes the system to the composite state "low-3, Low-B". If at this point the user decides to transition the system manually by pushing up, the composite state that is reached is "high-1, High," and all is consistent. If, however, the user had decided to upshift manually when the machine was still in

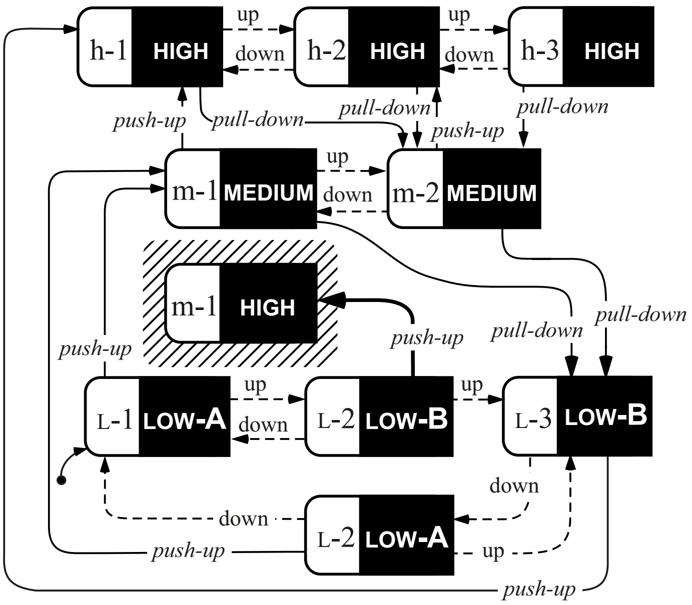

Figure 4. Composite of the machine and the user model.

state low-2, the machine would transition to state medium-1 (see Figure 1) and the interface would transition into High mode (see Figure 3). The new composite state would be "medium-1, High" (Figure 4) which is clearly an inconsistency. The display indicates to the user that he or she is in High mode, whereas in fact the underlying machine is in Medium (internal state medium-1).

The composite state "medium-1, High" constitutes an error state because the machine is in one specification class (Medium) and the user model is in another (High). Because of this discrepancy between the models with respect to the specification classes, the user model (of Figure 3) is not a correct abstraction of the underlying machine. Given such a display, nothing can be done to alleviate the fundamental problem; no additional training, better user manuals, procedures, or any other countermeasures will help. We therefore can conclude that the user model of Figure 3 is incorrect for the task.

GENERATING USER INTERFACES

The objective of the interface generation procedure is to derive a user model that is correct for the specified tasks – namely, one that is free of error, restricting, and augmenting states. A second requirement is that this user model must be succinct. The proposed methodology centers on a systematic method for reducing the machine model into a smaller model that still allows the user to perform correctly all the specified tasks (and such that the model cannot be reduced further). What follows is a description of an algorithmic procedure for the generation of user models. The detailed mathematical aspects of this algorithm are provided in Heymann and Degani (2002). Here we shall describe the underlying ideas and principles of the methodology and illustrate the procedure with the aid of examples – in particular, the already-familiar example of the transmission system.

Outline of the Algorithmic Approach

The algorithmic approach for generation of succinct user models and associated interfaces is based on the fact that not all the system's internal states need to be individually presented to the user. Specifically, two internal states need not be distinguished whenever (a) they belong to the same specification class, (b) each user-triggered event that is available and active in one of the states is also available and active in the other, and (c) starting from either of the two states and triggered by the same event sequence, the state pairs visited also satisfy Conditions a and b. Such state pairs that need not be distinguished by the user are referred to as *compatible.* Thus, the first step of the interface generation algorithm consists of finding all the compatible state pairs. From these pairs, all the (largest possible) sets of compatible states – called *maximal compatibles* – are then computed.

The next step of the algorithm consists of generating a reduced user model. The user model's states are composed of maximal compatible state sets, which constitute the user model's building blocks. In general, not all the maximal compatibles need to be chosen for the reduced model, and frequently the designer has more than one choice in selecting appropriate compatible sets. The key to a suitable selection is that the selected set must 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 selected maximal compatible (this constitutes the cover property). The state set is selected by first choosing maximal compatibles that constitute a minimal cover of the machine's state set (i.e., none of the selected maximal compatibles can be omitted from the selected set without violating the cover property). If necessary, additional maximal compatibles are incrementally added to the selected minimal cover so as to ensure that the set of target states of each transition emanating from a maximal compatible is included in some maximal compatible of the selected set. (In the worst case, this incremental addition of maximal compatibles will terminate when all maximal compatibles are chosen. In other words, the set of all maximal compatibles is the upper bound for the reduced user model state set.)

Once the state set of the reduced model has been selected as described, the next step is to determine the transitions in the reduced model. These are defined so as to be consistent with the original machine model and with the partition of the state set into specification classes. Three subtle issues arise and are dealt with in this connection: (a) sets of distinct event that need not be distinguished and can be grouped together, (b) events that can be deleted because their presence in the reduced model is redundant, and (c) transition nondeterminism that can be eliminated from the reduced model. The resultant reduced machine model constitutes the user model. The required interface is then extracted from this model.

Compatible States

The user model must enable the user to operate the machine correctly with respect to his or her tasks and requirements. Thus, although the user is required to track, unambiguously, the specification classes visited by the system, the user need not track every internal state of the machine. In the transmission example there is no need for users to distinguish between two internal states (say, medium-1 and medium-2 of the Medium mode) if, following any event sequence, (a) they always end up in the same specification class (e.g., High) and (b) the same set of user-triggered events is available, regardless of which of the two internal states they started from. If that is the case, the two states (medium-1 and medium-2) are compatible. From an interface design standpoint, the two compatible states can be grouped together on the display and represented as a single user model state because the intrinsic details of whether the current internal state is medium-1 or medium-2 are inconsequential to the user.

Instead of trying to find all state pairs that are compatible, it is computationally more convenient to first find all state pairs that are incompatible. Once all incompatible pairs (i.e., those that cannot be grouped together on the interface) are identified and marked, the remaining state pairs must be compatible.

An immediate criterion for incompatibility of a state pair is that the two states that constitute the pair belong to two distinct specification classes or have distinct active user-triggered events. For example, the state pair low-1 and high-1 is outright incompatible because low-1 belongs to the Low mode and High-1 belongs to the High mode. These two states must never be grouped together on the display.

The second criterion for designating a pair of states as incompatible has to do with event sequences. A state pair is marked as incompatible if, starting from the two states and following the same sequence of events, a transition is made into a state pair that has already been deemed incompatible. For example, consider the state pair low-2 and low-3. Initially, the pair is tentatively marked as compatible because the two states belong to the same specification class and have the same active user-triggered event (*push-up*). However, following a common event (*push-up*), this pair transitions into the state pair medium-1 and high-1. Because medium-1 and high-1 are already known to be incompatible (because they belong to two different specification classes), the initial pair, low-2 and low-3, must also be marked as incompatible.

Computing Compatible Pairs

An efficient iterative procedure for computing such compatible and incompatible state pairs is based on the use of *merger tables* (see Kohavi, 1978, and Paull & Unger, 1959), as we will describe using the transmission example. 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. By iteratively stepping through the table one event transition at a time, one can progressively detect all incompatible state pairs, thereby "resolving" the table; that is, one can uncover all the state pairs that are not found to be incompatible and thus designate them as compatible.

In the case of the transmission example, there are 8 states and $[n \times (n-1)/2] = [8 \times 7/2] = 28$ possible state pairs. Each state pair corresponds to a unique cell in the table.

Initial resolution. Figure 5 shows the merger table for the transmission system and its initial resolution. Based on the observations from the previous subsection regarding incompatible pairs, the following procedure is used to populate the cells:

1. For each state pair (e.g., low-1 and high-3) that can be immediately determined as incompatible, because they belong to two distinct specification classes (Low and High) or have distinct sets of active user-triggered events, the cell is marked as incompatible.
2. In the cells for all other state pairs, write in the next state pair or pairs that they transition into following a common event. For example, for the state pair medium-1 and medium-2, the next state pair, following the common event (*push-up*), is high-1 and high-2.

Beginning at the top of the table, the uppermost cell represents the state pair "low-1, low-2." Looking at the machine model (see the inset in Figure 5), note that states low-1 and low-2, transition on automatic upshift (*up*) to low-2 and low-3 – and that (low-2, low-3) is written inside the top cell. Next, go down to the cell representing the state

	low-1	low-2	low-3	medium-1	medium-2	high-1	high-2
low-2	$\begin{bmatrix}\textbf{low-2}\\ \textbf{low-3}\end{bmatrix}$						
low-3	$\begin{bmatrix}\textbf{medium-1}\\ \textbf{high-1}\end{bmatrix}$	$\begin{bmatrix}\textbf{medium-1}\\ \textbf{high-1}\end{bmatrix}$ $\begin{bmatrix}\textbf{low-1}\\ \textbf{low-2}\end{bmatrix}$					
medium-1	Incompatible	Incompatible	Incompatible				
medium-2	Incompatible	Incompatible	Incompatible	$\begin{bmatrix}\textbf{high-1}\\ \textbf{high-2}\end{bmatrix}$			
high-1	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		
high-2	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible	$\begin{bmatrix}\textbf{high-2}\\ \textbf{high-3}\end{bmatrix}$	
high-3	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		$\begin{bmatrix}\textbf{high-1}\\ \textbf{high-2}\end{bmatrix}$

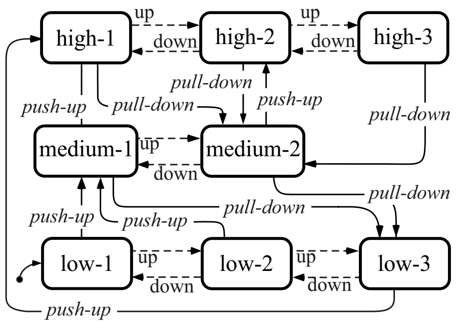

Figure 5. The merger table for the eight-state transmission system and its initial resolution.

pair “low-1, low-3.” Note that in the machine model these two states transition on manual upshift (*push-up*) into medium-1 and high-1 – and that’s what gets written inside the cell. Moving one cell to the right to the cell representing “low-2, low-3,” note that in the machine model there are two common transitions from this pair: an automatic downshift (event *down*) to low-1 and low-2 as well as a manual upshift (event *push-up*) to medium-1 and high-1 – these two state-pairs are therefore written inside the cell.

Now, go down the table to the cell representing “low-1, medium-1.” Since each state of this pair belongs to a different specification class, they are immediately deemed incompatible. The same applies for low-1 and medium-2. In this fashion, it is possible to go cell by cell and populate the rest of the table. Notice, however, that the cell representing “high-1, high-3” is empty. This is because these two states are not incompatible (they both belong to the High mode and have *push-down* as an active user-triggered event), yet they don’t transition into another state pair under a common event like the rest of the state pairs. Therefore the cell is left empty, and will be dealt with later.

Second iteration. We can now continue with the resolution process, but from this step onward it is not necessary to refer to the machine model anymore. In an iterative manner, start substituting state pairs in the cells according to the following procedure:

1. Cells that were already marked as incompatible stay that way.
2. Every cell that has not yet been determined as incompatible in Figure 5 (e.g., “low-1, low-3”) is updated as follows: If a cell includes a state pair (e.g., medium-1 and high-1) that has already been marked as incompatible, then the cell is designated incompatible (see Figure 6).
3. Otherwise, the cell is modified as follows: Each state pair in the cell is replaced by all the state pairs that appeared in their original cell. For example, in Figure 5 the cell representing “low-1, low-2” contains the pair low-2 and low-3. Look into the cell representing “low-2, low-3” in Figure 5 and note the two state pairs: “low-1, low-2” and “medium-1, high-1.” Write these two state pairs inside the cell representing “low-1, low-2” (in Figure 6).

low-2	$\begin{bmatrix}\text{medium-1}\\ \text{high-1}\end{bmatrix}$ $\begin{bmatrix}\text{low-1}\\ \text{low-2}\end{bmatrix}$						
low-3	Incompatible	Incompatible					
medium-1	Incompatible	Incompatible	Incompatible				
medium-2	Incompatible	Incompatible	Incompatible	$\begin{bmatrix}\text{high-2}\\ \text{high-3}\end{bmatrix}$			
high-1	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		
high-2	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible	$\begin{bmatrix}\text{high-1}\\ \text{high-2}\end{bmatrix}$	
high-3	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		$\begin{bmatrix}\text{high-2}\\ \text{high-3}\end{bmatrix}$
	low-1	**low-2**	**low-3**	**medium-1**	**medium-2**	**high-1**	**high-2**

Figure 6. The second iterative resolution.

Continuing with the procedure, the cell representing "low-2, low-3" in Figure 6 is now designated as incompatible (because it contains the pair medium-1 and high-1, which was already marked as incompatible). In the cell representing "medium-1, medium-2," place the state pair high-2 and high-3. The cell representing "high-1, high-2" gets the state pair high-1 and high-2, and the cell for "high-2, high-3" gets the pair high-2 and high-3, whereas the cell "high-1, high-3" stays empty, as before.

Third iteration. In the next iteration the table of Figure 7 is obtained. Here the cell representing "low-1, low-2" is marked incompatible (because it contains medium-1 and high-1).

Final iteration. In this step, because no additional incompatible pairs are identified, the table remains identical to that of Figure 7. From here on, no further iterations will ever produce incompatible pairs. Therefore, the empty cell representing "high-1, high-3" is marked as compatible, concluding the resolution procedure.

The resolution procedure identified all the incompatible and compatible pairs. Figure 8 shows that there are four such compatible pairs:

(high-1, high-2), (high-1, high-3), (high-2, high-3)
(medium-1, medium-2)

What this means is that when it comes to designing the interface for the transmission system, it will be possible to combine a compatible pair (e.g., medium-1, medium-2) into a single indication (because the user does not need to distinguish between medium-1 and medium-2 in order to perform the task). Notice, however, that the states low-1, low-2, and low-3 do not appear in any compatible pairs. As a consequence, no reduction can be achieved with respect to these three internal states.

Identifying Compatible Sets

Although all the compatible pairs in the system have been identified and the pairs can be combined so as to reduce the number of states (and corresponding display indications) to create an abstracted user model, the process is not yet finished

	low-1	low-2	low-3	medium-1	medium-2	high-1	high-2
low-2	Incompatible						
low-3	Incompatible	Incompatible					
medium-1	Incompatible	Incompatible	Incompatible				
medium-2	Incompatible	Incompatible	Incompatible	[high-2 high-3]			
high-1	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		
high-2	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible	[high-1 high-2]	
high-3	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		[high-2 high-3]

Figure 7. The third iterative resolution.

because it may be possible to further reduce the system by considering compatible triples, quadruples, and so forth. The idea here is based on the observation that a set of states is compatible if all its constituent state pairs are compatible. That is, a *state triple* is compatible if its three constituent pairs are compatible; a *state quadruple* is compatible if its four constituent triples are compatible; and so on. Recall that the goal is to try to reduce the system as much as possible; the larger the compatible sets (or the larger the compatible *n*-tuples), the better. Thus, this reduction procedure allows one to find the maximal compatibles.

Returning to the transmission example, note that the set of compatible pairs (high-1, high-2; high-1, high-3; and high-2, high-3) constitutes a compatible triple. What this means is that it is possible to combine high-1, high-2, and high-3 into a single indication on the interface. In principle, after finding this compatible triple, the automated procedure will try to find bigger compatible sets (i.e., a quadruple in this case). However, a triplet is the best that can be done with the transmission system, and the procedure terminates with the following set of maximal compatibles:

1. (high-1, high-2, high-3);
2. (medium-1, medium-2);
3. (low-1), (low-2), (low-3).

Constructing the Reduced User Model

The set of maximal compatibles forms the basis from which the user model is constructed. The internal states high-1, high-2, and high-3 are combined into a single state called "high," and medium-1 and medium-2 into a state called "medium." Separate states are necessary for low-1, low-2, and low-3. The reduced user model obtained for the transmission system is shown in Figure 9 where the states are depicted as modes High, Medium, Low-A, Low-B, and Low-C.

Note that both the Medium and High modes have self-loops. These are the internal events that take place "inside" Medium and High. Since these internal (machine-triggered) events do not cause any changes in the user model (i.e., there is no mode switching), it is possible to go ahead and

	low-1	low-2	low-3	medium-1	medium-2	high-1	high-2
low-2	Incompatible						
low-3	Incompatible	Incompatible					
medium-1	Incompatible	Incompatible	Incompatible				
medium-2	Incompatible	Incompatible	Incompatible	**Compatible**			
high-1	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible		
high-2	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible	**Compatible**	
high-3	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible	**Compatible**	**Compatible**

Figure 8. The final resolution.

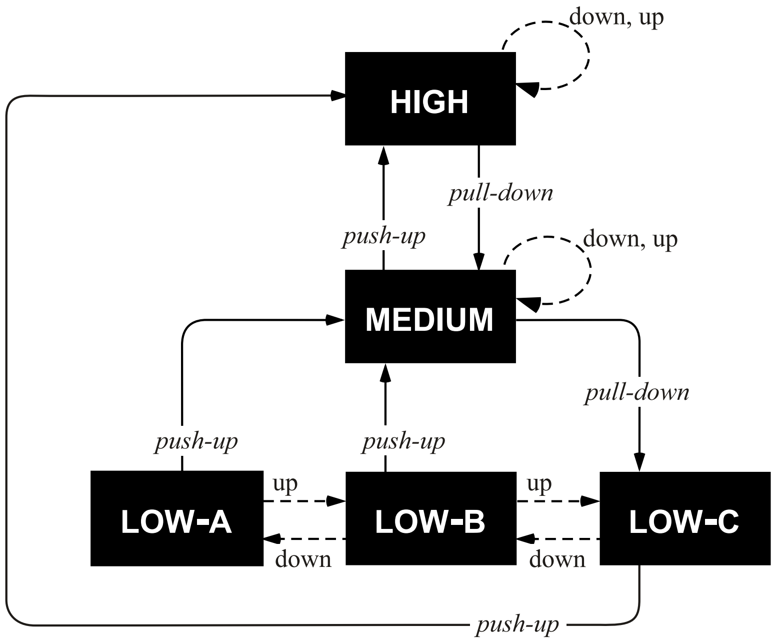

Figure 9. Succinct and correct user model for the transmission system.

delete them. Nevertheless, beyond the results of the formal procedure, it is up to the design team to decide, based on operational and situation awareness consideration, whether they want to provide some annunciation (e.g., blinking) about their occurrence, on the interface. It is noteworthy, however, that in the case of automatic transmission systems, most car manufacturers opt not to provide any indication to the user about internal gear shifts while the car is in Drive mode.

At this point, it is possible to evaluate the correctness of the resulting user model by employing the verification procedure mentioned earlier in the paper – making sure that no errors have crept in while constructing the interface or anywhere throughout the process.

Specifying the Interface and the User Manual

The final step is to extract from the user model all the (state) information that must be provided on the display and then specify the indications and all the (event related) information that must appear in the user manual. The abstraction process of the transmission system is now complete – it has been reduced as much as possible, providing the basis for a correct and succinct display (as well as all the necessary information for the user manual).

FURTHER ASPECTS OF THE REDUCTION PROCEDURE

The transmission system that we used to illustrate the reduction procedure was selected because of its familiarity and its limited number of states and transitions. As a consequence, not all aspects of the algorithmic approach could be exhibited. Next we shall present another, more complex, example that will exhibit further aspects of the reduction procedure.

The machine in Figure 10 has 18 states and 42 transitions (some of which are user triggered, such as *ua* and *ud;* the rest are automatic). Four specification classes are defined for this machine: Classes A, B, C, and D. The task requirement is similar to the previous one: The user must be able to identify the current specification class (mode) of the machine and to anticipate the next mode that the machine will enter as a result of his or her interactions.

The reduction procedure is performed as described in the previous section. The algorithm terminates with the following list of eight maximal compatibles:

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).

Note that unlike the transmission system, in which each internal state appeared in only one maximal compatible, this example illustrates a case in which there are multiple overlapping compatible sets. In particular, the first four maximal compatibles contain the states 21, 22, 31, and 32. This overlap among maximal compatibles is quite common and frequently implies the existence of multiple candidate user models. In the example, there are two candidates to choose from: One consists of the compatibles 1, 4, 5, 6, 7, 8 as its state set, and the other consists of 2, 3, 5, 6, 7, 8. (These two sets constitute the only possible minimal covers, as discussed earlier, in the section titled Outline of the Algorithmic Approach.)

The selection among the various candidate user models cannot, generally, be quantified and is based on engineering and human factors considerations. Here various kinds of design decisions can be brought to bear: The number of user model states, the number (and intuitive nature) of the displayed transitions, the physical interpretation of the reduced model, and so forth. Of course, when no profound reason exists to prefer one candidate model over another, any one may be selected. In the present example we selected the minimal cover that consists of the maximal compatibles 2, 3, 5, 6, 7, 8 for reasons that will become clear later.

To construct the user model, the selected maximal compatibles are incorporated into user model states (modes A-1 and A-2; B; C-1 and C-2; and D). Next, the transitions between the user model states are established in the following way: For each user model mode and each event label that emanates from it, we mark the set of all constituent machine model target states. Figure 11 is a list of all user model modes, their respective event labels, and their resultant machine model target states. Finally, a transition with the corresponding label is drawn from the mode under consideration to each mode that includes *all* the machine model

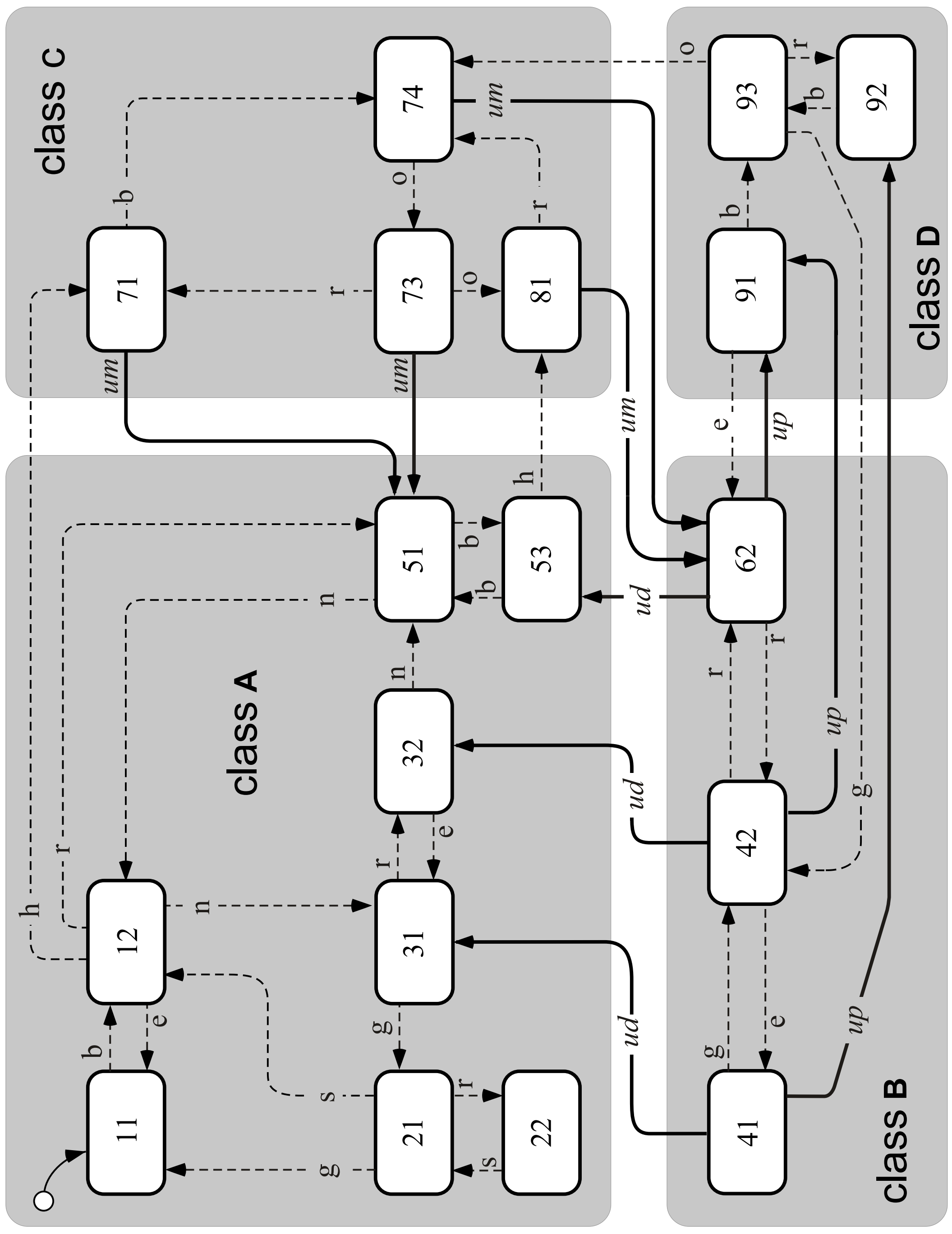

Figure 10. Machine model.

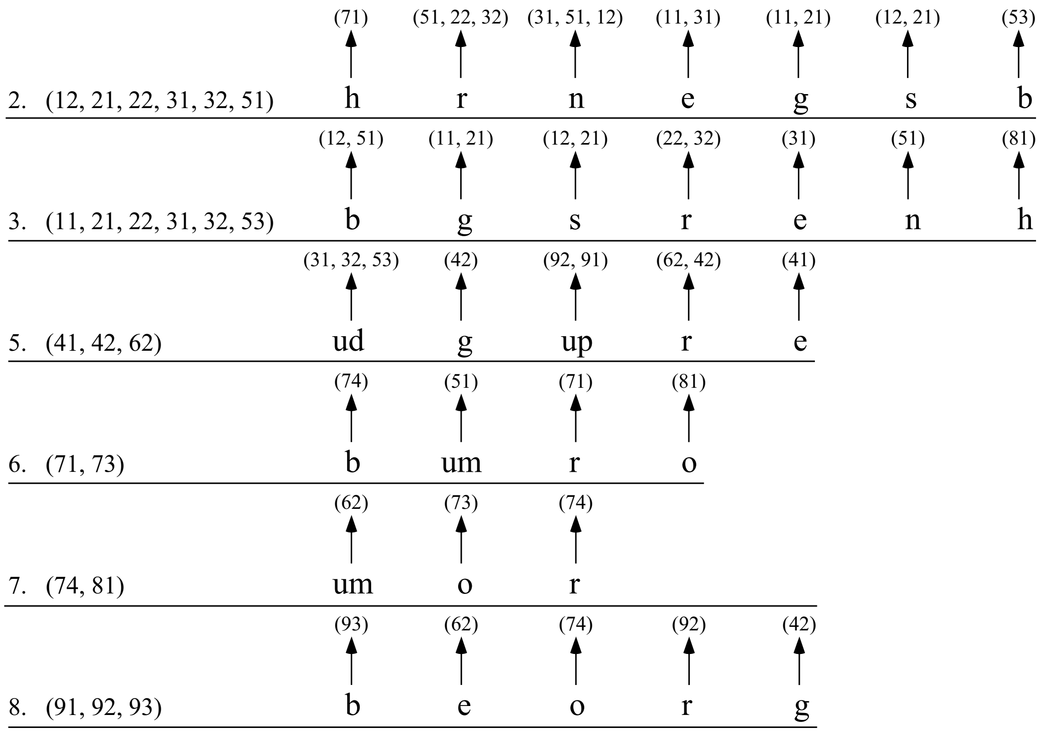

Figure 11. List of all user model modes, their respective event labels, and resultant machine model target states.

target states. When applied to all user model modes, this procedure results in the reduced model shown in Figure 12.

Note that there may be nondeterministic outgoing transitions to states within the given specification class. (This nondeterminism does not lead to nondeterministic transitions *between* specification classes and, hence, cannot lead to error states.) For example, the event *r* emanates from mode A-2 both to A-1 and, as a self-loop, to A-2. This nondeterminism can be eliminated by judicious decision as to which of the redundant transitions to delete. Note further that automatic events that occur *only* in self-loops have no effect on the reduced model and can be deleted. Thus, when the redundant transition *r* from A-2 to A-1 is deleted, the event *r* remains only in self-loops.

Finally, groups of events that *always* appear together in transitions can be abstracted into single representative labels. Thus, in the example, the events *n* and *s* are abstracted into the representative label *p,* and the events *e* and *g* are abstracted into *q*. The resulting user model, which is both correct and succinct, is depicted in Figure 13. It contains only six modes.

In addition to the six indicated modes, the user would need to know which user events can be triggered and what will be the ensuing mode. Thus, in mode B the user can trigger either *ud* or *up,* leading the system to A-2 or to D, respectively. In C-1 and C-2 the user can trigger event *um,* leading the system to modes A-1 or B.

Finally, recall that the reduced model of Figure 12 and the user model of Figure 13 correspond to the candidate consisting of maximal compatibles 2, 3, 5, 6, 7, 8. In this case the models constitute a minimal cover. Had we chosen the second candidate (consisting of maximal compatibles 1, 4, 5, 6, 7, 8) as the basis for our user model, we would have had to increment with maximal compatible 2 (as discussed in the section titled Outline of the Algorithmic Approach). The resulting user model (consisting of maximal compatibles 1, 2, 4, 5, 6, 7, 8) is not a minimal cover, but it is still an irreducible model.

SUMMARY AND CONCLUSIONS

We began this paper with a discussion on a formal approach for describing and analyzing human-automation interaction. Two objectives guided us: The first and foremost was that the user model and

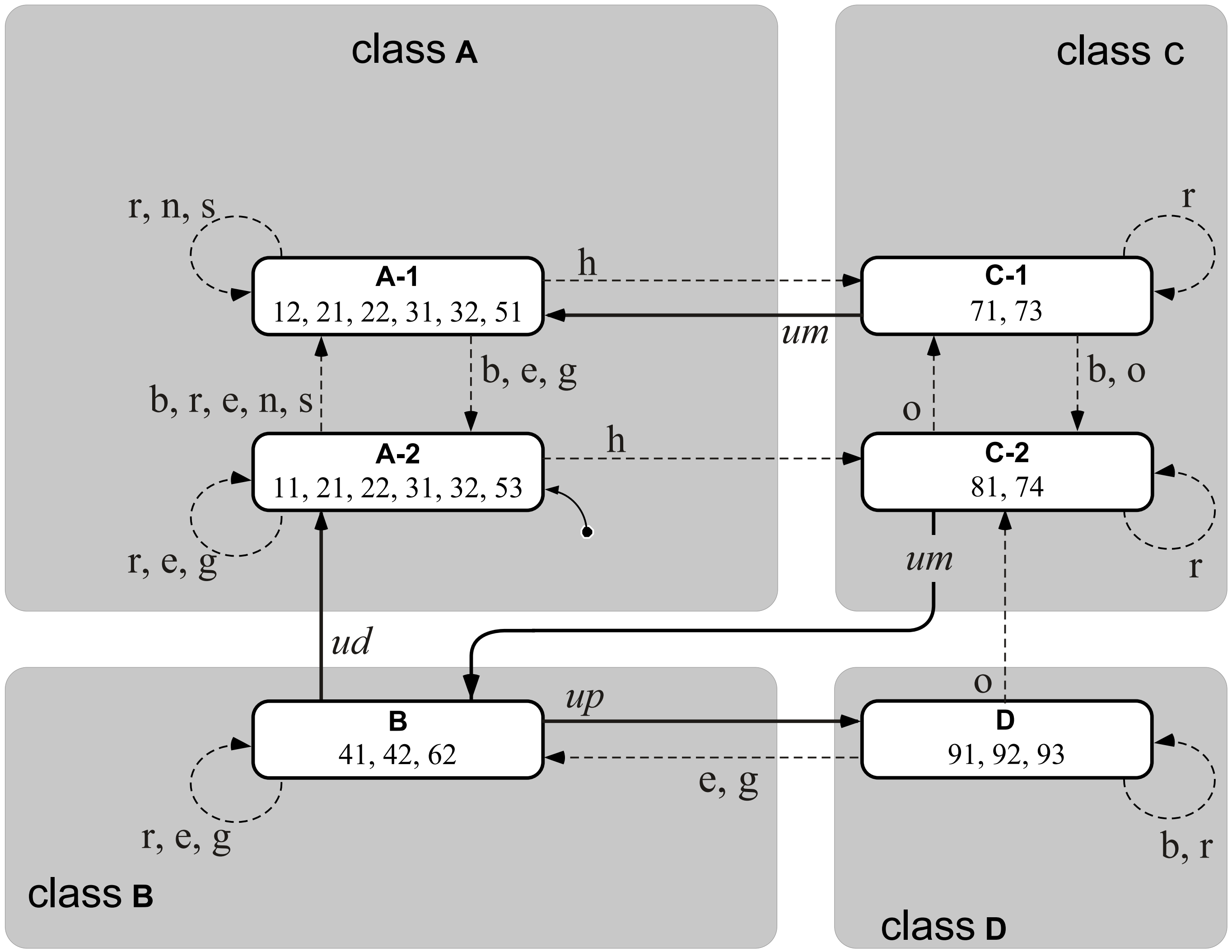

Figure 12. Reduced machine model.

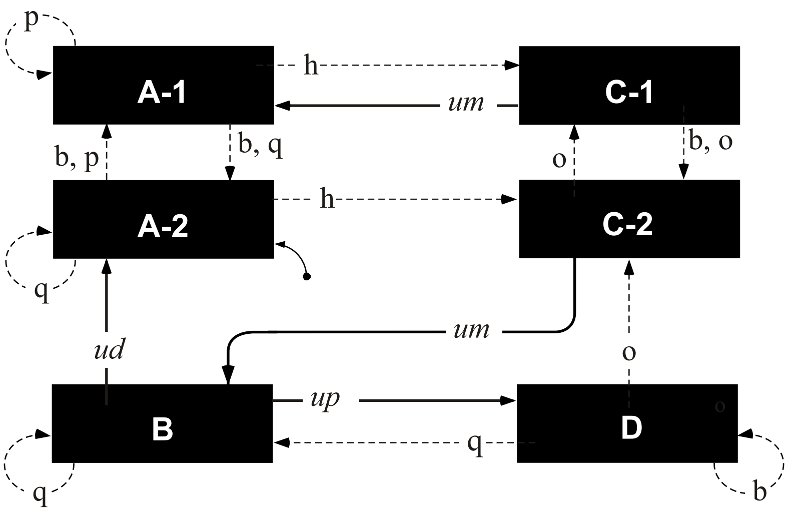

Figure 13. Succinct and correct user model and interface.

interface were correct; the second was that they were minimal, or succinct, in terms of the amount of information (e.g., mode annunciations, selection buttons, parameter settings, and user manual content) required to accomplish the task. We then focused our attention on a systematic procedure for reducing the machine model according to the user's task. The reduction algorithm described in this paper generates user models that are both correct and succinct.

Limitations

To analyze and generate user models according to the methodologies described in this paper, one needs a formal description of the underlying machine, specification classes, and task requirements. Although the use of such formal descriptions is currently not the mainstream in human factors, formal descriptions of system behavior and requirement specifications are used in many software development processes (e.g., the Unified Modeling Language methodology). Furthermore, many tools are now available that allow designers to specify the system's behavior (see Harel & Politi, 1998), and then the tool automatically translates the specification into code (e.g., Java or C++). We believe that just as software design is moving toward the use of formal methods for specification, design, and verification, interface design will eventually follow suit.

For simplicity and clarity of exposition, we have confined our discussion to machine models, specification classes, and task requirements that are based on discrete events and modeled as state transition systems. Nevertheless, the focus of this work is not on a particular modeling formalism and notation. Rather, it is on the ideas that they encapsulate. As such, the approach, methodology, and algorithm proposed here can be extended to other discrete event formalisms, such as Petri nets and Statecharts, as well as to hybrid systems models that have both continuous and discrete behaviors (e.g., see the hybrid system modeling and verification approach used in Oishi, Tomlin, & Degani, 2003).

In principle, the computation of maximal compatibles for very large systems with thousands of states can become exponentially complex and, eventually, computationally intractable. Nevertheless, many algorithmic techniques deal with this problem (e.g., Kam, Villa, Brayton, & Sangiovanni-Vincentelli, 1997). Using a computerized tool (Shiffman, Degani, & Heymann, 2005), the reduction algorithm described in this paper has been successfully applied to machine models with more than 500 internal states. It may be possible, by improving the efficiency of the algorithm, to reduce even larger machines.

Implications for Design of User Interaction

Most users perceive the interface as if it were the machine itself. On one hand, this induced misconception is an important design goal (e.g., "direct manipulation"), providing a smooth, effortless, and nonintermediary human-machine interaction. Although it is debatable whether or not it is good to always furnish this perception, it is obligatory that user interface designers not succumb to this illusion, which, unless carefully designed and verified, can backfire. For example, if there is a design flaw in the interface such that the delicate synchronization between the interface and the machine is disrupted, the interface may give the impression that the machine is doing one thing when in fact it is doing something completely different. In consumer electronics, Internet applications, and information systems this type of design flaw leads to user confusion and frustration. In high-risk systems, it can be disastrous.

Our discussion and the transmission example illustrate that even for machines that are seemingly simple (i.e., with relatively few states and straightforward user interaction), coming up with a correct and succinct interface is not a trivial matter. Interfaces that intuitively may appear to be correct have been shown, after applying formal verification, to be incorrect. While many interface correctness problems are indeed observed in simulations and usability testing, some are left unidentified and can plague a system for years. As systems become larger and more integrated (i.e., comprising several subsystems that are linked and synchronized), it becomes more difficult to evaluate user interfaces using traditional inspection-based methods. At the same time, there is an ever-increasing demand for reliable and safer user interaction.

Beyond incorrect interfaces, there exists the related issue of succinct interfaces. Given the current intuitive and iterative approach for generating design solutions, there is never a guarantee that the selected interface solution cannot be further reduced. It may very well be the case that some current interfaces are more complicated (e.g., require lengthy interaction sequences) than is necessary. To this end, we hope that the notion of abstraction – which is at the cornerstone of our formal approach for interface design and evaluation – as well as the interface correctness criteria, verification methodology, and procedure for generating correct interfaces will help interface designers to better understand and reason about critical design issues that are currently addressed in an intuitive and ad hoc way.

ACKNOWLEDGMENTS

The work described in this paper was funded by NASA's basic research and technology effort on human-automation interaction as well as the Integrated Intelligent Flight Deck project, Aeronautics Research Mission Directorate. The first author was supported by Grant NCC 2-798 from the NASA Ames Research Center to the San Jose State University. Michael Shafto and Kevin Jordan provided much-needed support and encouragement for this research work. Ronen Erez wrote the first version of the model reduction software. Smadar Shiffman contributed to this research effort by modifying and improving the algorithm described in this paper and by building a Web-based tool for verification and generation of interfaces.

REFERENCES

Bauer, B. (1996). Generating user interface from formal specifications of the application. In J. Vanderdonckt (Ed.), *Proceedings of the 1996 Computer-Aided Design of User Interfaces Conference* (pp. 141–152). Namur, Belgium: Presses Universitaires de Namur.

Bösser, T., & Melchoir, E. M. (1992). The SANE toolkit for cognitive modelling and user-centered design. In M. Galer, S. Harker, & J. Ziegler (Eds.), *Methods and Tools in User-Centred Design for Information Technology* (pp. 93–126). Amsterdam: North-Holland.

Browne, T., Davila, D., Rugaber, S., & Stirewalt, K. (1997). Using declarative descriptions to model user interfaces with MASTERMIND. In F. Paternò & P. Palanque (Eds.), *Formal methods in human-computer interaction* (pp. 93–120). London: Springer-Verlag.

Degani, A. (1996). *Modeling human-machine systems: On modes, error, and patterns of interaction.* Unpublished doctoral dissertation, Georgia Institute of Technology, Atlanta.

Degani, A. (2004). *Taming HAL: Designing interfaces beyond 2001.* New York: Palgrave Macmillan.

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

Degani, A., Heymann, M., & Shafto, M. (1999). Formal aspects of procedures: The problem of sequential correctness. In *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting* (pp. 1113–1117). Santa Monica, CA: Human Factors and Ergonomics Society.

Dix, A. J. (1991). *Formal methods for interactive systems.* London: Academic Press.

Doherty, G. J., Campos, J. C., & Harrison, M. D. (2000) Representational reasoning and verification. *Formal Aspects of Computing, 3,* 260–277.

Duke, D. J., Fields, B., & Harrison, M. D. (1999). A case study in the specification and analysis of design alternatives for a user interface. *Formal Aspects of Computing, 11,* 107–131.

Foley, J. D., & Wallace, V. L. (1974). The art of natural graphic man-machine conversation. *Proceedings of the IEEE, 62,* 462–471.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming, 8,* 231–274.

Harel, D., & Politi, M. (1998). *Modeling reactive systems with Statecharts: The STATEMATE approach.* New York: McGraw-Hill.

Harrison, M., & Thimbleby, H. (1990). *Formal methods in human-computer interaction.* Cambridge, UK: Cambridge University Press.

Heymann, M., & Degani, A. (2002). *On abstractions and simplifications in the design of human-automation interfaces* (NASA Tech. Memorandum 2002-211397). Moffett Field, CA: NASA Ames Research Center.

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

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

Kam, T., Villa, T., Brayton, R., & Sangiovanni-Vincentelli, A. (1997). Implicit computation of compatible sets for state minimization of

ISFSM's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16,* 657–676

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22,* 365–394.

Kohavi, Z. (1978). *Switching and finite automata theory.* New York: McGraw-Hill.

Krzystrof, G., & Weld, D. (2004). SUPPLE: Automatically generating user interfaces. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces* (pp. 93–100). New York: Association for Computing Machinery.

Leveson, N. (1995). *Safeware: System safety and computers*. New York: Addison-Wesley.

National Transportation Safety Board. (1997). *Grounding of the Panamanian passenger ship Royal Majesty on Rose and Crown shoal near Nantucket, Massachusetts on June 10, 1995.* Washington, DC: Author. (NTIS No. PB97-916401)

Norman, D. (1983). Design rules based on analysis of human error. *Communications of the ACM, 26,* 254–258.

Norman, D. (2004). *Emotional design*. Cambridge, MA: Basic Books.

Oishi, M., Tomlin, C., & Degani, A. (2003). *Discrete abstraction of hybrid systems: Verification of safety and application to user-interfaces* (NASA Tech. Memorandum 212803). Moffett Field, CA: NASA Ames Research Center.

Palanque, P., & Paternò, F. (1998). *Formal methods in human computer interaction*. London: Springer-Verlag.

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2000). A model for the types and levels of human interaction with automation. *IEEE Transaction on Systems, Man, and Cybernetics – Part A: Systems and Humans, 30,* 286–297.

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

Paternò, F., & Santoro, C. (2002). Preventing user errors by systematic analysis of deviations from the system task model. *International Journal of Human-Computer Studies, 56,* 225–245.

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, EC-8,* 356–367.

Rodriguez, M., Zimmerman, M., Katahira, M., de Villepin, M., Ingram, B., & Leveson, N. (2000, October). Identifying mode confusion potential in software design. Paper presented at the *19th Digital Aviation Systems Conference* held in Philadelphia, Pennsylvania.

Rushby, J. (1999, June). *Using model checking to help discover mode confusions and other automation surprises.* Presented at the Third Workshop on Human Error, Safety, and System Development, Liege, Belgium.

Rushby, J. (2001). Analyzing cockpit interfaces using formal methods. *Electronic Notes in Theoretical Computer Science, 43*(May), 1–14.

Shiffman, S., Degani, A., & Heymann, M. (2005). UIverify – A Web-based tool for verification and automatic generation of user interfaces. In *Proceedings of the 8th Annual Applied Ergonomics Conference* [CD-ROM]. Norcross, GA: Institute of Industrial Engineers.

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, E., & Slacher, E. (1996). Declarative interface models for user interface construction tools: The MASTERMIND approach. In L. Bass, & C. Unger (Eds.), Engineering for Human-Computer Interaction, *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction* (pp. 120–150). London: Chapman & Hall.

Thimbleby, H., Blandford, A., Cairns, P., Curzon, P., & Jones, M. (2002). User interface design as systems design. In X. Faulkner, J. Finlay, & F. Détienne (Eds.), *Proceedings of People and Computers XVI Conference* (pp. 281–301). London: Springer-Verlag.

Wasserman, A. I. (1985). Extending state transition diagrams for the specification of human-computer interaction. *IEEE Transactions on Software Engineering, 11,* 699–713.

Michael Heymann is a professor of computer science and director of Center for Intelligent Systems, holding the Carl Fechheimer Chair in Electrical Engineering, at the Technion, Israel Institute of Technology. He received his Ph.D. in chemical engineering in 1965 from the University of Oklahoma, Norman.

Asaf Degani is a research scientist in the Computational Sciences Division, NASA Ames Research Center, Mountain View, California. He received his Ph.D. in industrial and systems engineering from the Georgia Institute of Technology in 1996.

Date received: August 13, 2003
Date accepted: December 23, 2005