Paper accepted for publication, *Human Factors Journal*. (expected date of publication – 2007)

FORMAL ANALYSIS AND AUTOMATIC GENERATION OF USER INTERFACES: APPROACH, METHODOLOGY, AND AN ALGORITHM

Michael Heymann
Department of Computer Science
Technion, Israel Institute of Technology

Asaf Degani
NASA Ames Research Center
Moffett Field, CA

ABSTRACT

In this paper we propose a formal approach and methodology for analysis and generation of human-machine interfaces, with special emphasis on human-automation interaction. Our approach focuses on the information content of the interface—that is, on "what should be presented"—rather than on the form and layout of the graphical user interface ("how it should be presented"). The methodology is guided by two criteria: First and foremost, the interface must be correct. That is, given the interface indications and all related information (e.g., user manuals, training material, etc.), the user must be able to successfully perform the specified tasks. Second, the interface and related information should be succinct—that is, the amount of information (e.g., mode indications, mode buttons, parameter settings, etc.) presented to the user should be reduced (abstracted) to the minimum necessary. The foundation of our approach is the notion of information abstraction. We argue that in terms of the information provided on the interface, user interfaces are always an abstract description of the underlying behavior of the machine. The correspondence between what is presented to the user and the inner working of the machine is the focus of the verification and abstraction method presented in this paper. We discuss these concepts and illustrate a step-by-step procedure for generating correct and succinct user interfaces. Two examples are used to illustrate the procedure. The procedure for generating interfaces can be automated, and a software system for its implementation has already been developed.

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 a must, 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, let us 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 feet. The pilot enters the 10,000-feet altitude constraint into the autopilot, engages a mode called "VERTICAL SPEED," then selects the rate of climb (e.g., 2,000 feet per minute); now the aircraft begins climbing to 10,000 feet. 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 per minute) 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. Under 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. What we have here is that 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. We say that such an interface is "incorrect" because the pilots, in the process of performing a specified task (climb and level-off), cannot anticipate the consequences of their interaction with the machine. (See Degani and Heymann, 2002, for a detailed analysis of this particular interface problem and its root causes).

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 of interfaces is to make sure that the interface is indeed correct. Currently, this abstraction is performed in a heuristic and intuition-based manner, and 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, it also involves a complicated certification process (see for example Federal Aviation Regulation 25.1329 and associated Advisory Circular). 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 20 examples).

On the flip side of the abstraction problem lies the case where the interface provides *too much* information and overloads the user with superfluous and irrelevant information. Naturally, we all strive for interfaces (and user manuals) that are not only correct, but succinct. In general, we want an interface in which the number of modes, states, events, and parameters that the user needs to monitor and interact with to be *minimal*. In most cases we would prefer to have a small set of modes rather then a large set of modes in order to perform a given task (Norman, 1983). Likewise, we would prefer short sequences of user inputs rather than 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 et al., 2002). The advantage of having succinct displays and shorter sequences of user inputs is not only in minimizing the actual size of the user interface and the amount of indications that need to be designed and implemented, but also in reducing the perceptual and cognitive burden on the user.

A formal approach to human-computer interaction and a literature review

Many aspects of the 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, by focusing on the information content of the interface (rather than its appearance and layout) the problem of what to display and related user interaction issues 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 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 & Wallace (1974) and Jacob (1983) used similar modeling formalisms for developing general interface design specifications for human-computer interaction. 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 Human Computer Interaction (HCI) began using formal methods as a way to analyze and measure user interaction. Kieras & Polson (1985) used formal methods to quantify the complexity of human-computer interaction. To do this, they modeled both the device and the user's tasks as finite state machines. Since 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. Bosser & Melchoir (1990) 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, & Harrison (1999) describe 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, & 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, et al. (1995) 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, et al. (1997) used the MASTERMIND framework to develop an approach for automatically rendering user interfaces (e.g., dialogues 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 & Weld (2004) used an optimization algorithm for automatically selecting, resizing, and rendering of display widgets to accommodate different display sizes (e.g., small cell phones, PDAs, 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 here.

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: (1) the machine, (2) the user's tasks, (3) the user interface, and (4) the user's model of the machine.

Machine

We consider machines that interact with their human users, interact with 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, while 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 conditioner 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 that *user-triggered* transitions are depicted as solid lines, *while 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.

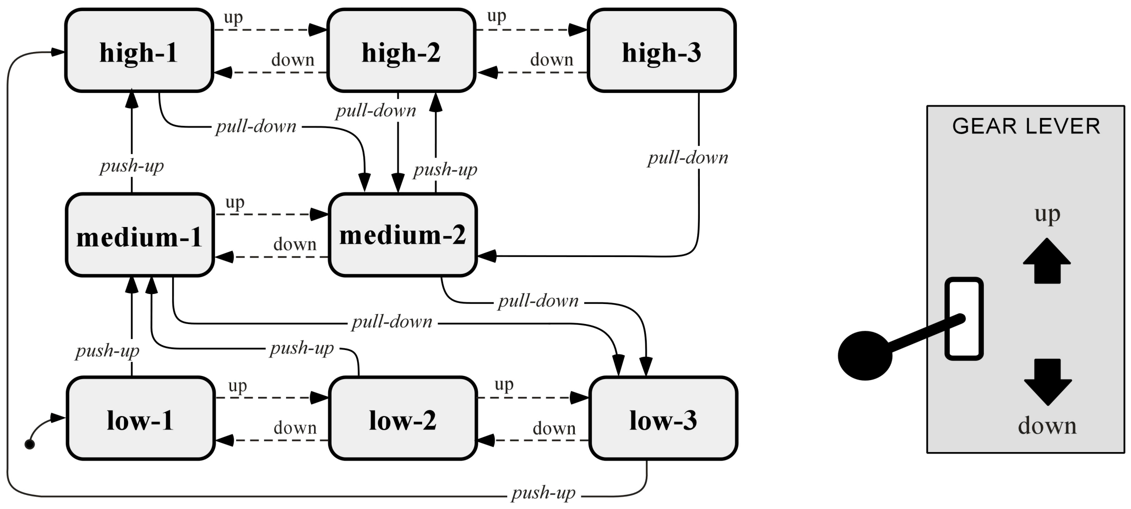


Figure 1. Transmission system of a vehicle (and the driver's gear lever)

The transmission system in Figure 1 has eight states. These states are grouped into three clusters, that we refer to as modes: LOW, MEDIUM, and HIGH. Thus there are internal "speed-level" states **low-1**, **low-2**, and **low-3** in the LOW mode; **medium-1** and **medium-2** in the MEDIUM mode; and **high-1**, **high-2**, and **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 up-shifts (to higher speed states) are denoted by the event *up*, and automatic down-shifts (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 torque level. These user-triggered transitions are denoted by events *push-up* and *pull-down*, respectively.

User's tasks

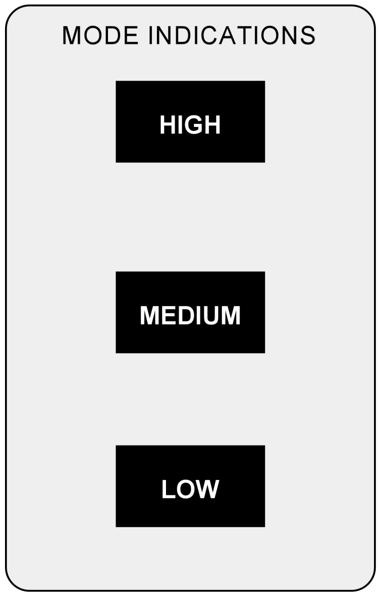
Generally speaking, users interact with a machine to achieve a specific set of tasks (Parssuramann et al., 2000). These tasks may vary widely, and range from using consumer electronic devices, such as VCRs, to interacting with web browsers and operating safety-critical systems (e.g., medical devices and navigation systems onboard 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. We do this 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 which the design team determined that the user need not distinguish among. For example, in the transmission system the three modes—LOW, LOW, MEDIUM, and HIGH—are our 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 classes. 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 all contained in HIGH do not need to be tracked). Using this approach, other types of user's tasks, such as reliably executing a specified sequence of actions, can also be expressed formally (Degani, Heymann, & Shafto, 1999).

Interface

In almost every machine that we encounter today, the events that take place inside the machine are purposefully abstracted and the interface displays only a limited view of these internal states. In most computer systems, there is dedicated software that 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, collocated between the system and the interface, takes in selected events from the machine and provides outputs in a form of commands to light up (or turn off) display indications. One important 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, let us return to the vehicle transmission example. Figure 2(a) 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 2(a) with Figure 1, the LOW mode has actually three internal states: **low-1**, **low-2**, and **low-3**. When the user first enters manually into low gear, **low-1** is the active state (see the small quarter-circle arrow); 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.



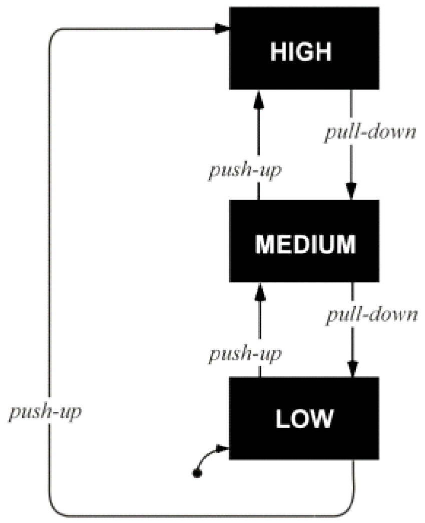


Figure 2(a). Proposed interface

Figure 2(b). The corresponding user model

User model

Manufacturers normally provide users with information about the working of the machine by means of user manuals. Here the manufacturers 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 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 2(a). 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, while a down-shift would transition the system to LOW mode, and so on. This series of fragmented statements describes to the user how the machine works, as well as how he or she is expected to interact with it. (But again, note that these user manual statements are abstractions of the actual behavior of the machine).

In Figure 2(b) 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, 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 it, 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 as can be readily seen in Figure 2(b), 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. Finally, it is important to note here that the description of the user model may not be confined to exactly what is provided in the user manual and/or on the interface, as is the case in this example.

INTERFACE CORRECTNESS CRITERIA

Among the four elements that play a role in the formal aspect of human-machine interaction, the *machine model* and the *user's tasks* are regarded for our purposes as given. (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 which is embedded in it) as the focus of our 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., up-shifting the gear) will sometimes transition the system into one mode (e.g., MEDIUM) and at other times into another (e.g., HIGH).

Broadly speaking, there are three user-interface correctness criteria that we aim at satisfying 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*, as explained below.

- An *error state* occurs when the user interface indicates that the machine 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 non-deterministic 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).

Non-deterministic interfaces and error states

We begin by analyzing the proposed user model of Figure 2(b). The manual up-shift from MEDIUM to HIGH and the down-shift 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: up-shifts from **low-1** and **low-2** take us to MEDIUM, while the up-shift from **low-3** switches the transmission to HIGH (see Figure 1.) What we have here is that the same user-triggered event (*push-up*) takes us to either of two different machine modes. But since the display abstracts from us which internal state the system is in, we will not be able to predict whether the system will transition to MEDIUM or HIGH. In other words, the proposed display, with respect to user-triggered events, becomes non-deterministic and may lead to an error state. Therefore, we must conclude that the proposed interface and corresponding user model of Figure 2 is incorrect.

Next consider the alternate user-model of Figure 3 where the display has been modified to partition the low mode into two sub-modes (LOW-A, and LOW-B). The user manual is modified correspondingly to explain to the user that the up-shift (*push-up*) from LOW-A, transitions the system to MEDIUM, while the up-shift (*push-up*) from LOW-B, transitions the system to HIGH. Let us try to analyze the correctness of this user model. But this time let us proceed in a formal way (Degani & Heymann, 2002).

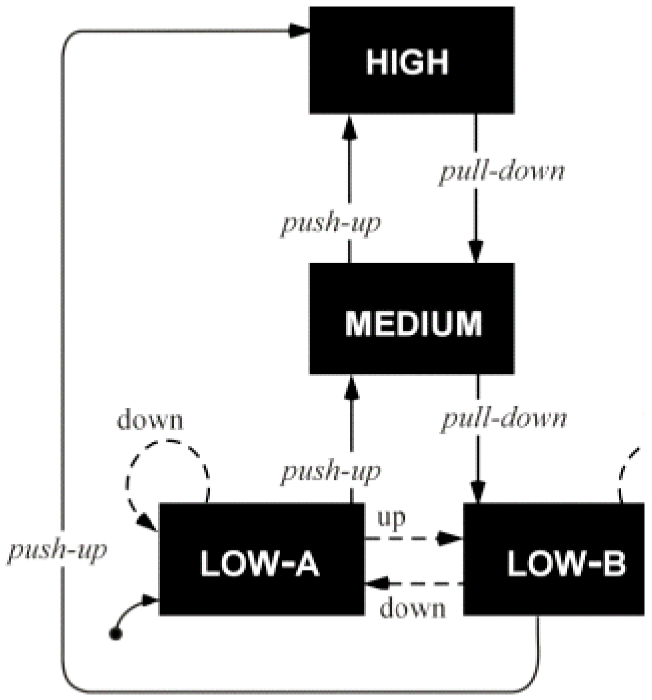


Figure 3. Alternative user model

In any human-machine system, two concurrent processes are constantly at play: the machine with its internal states and transitions on one hand, and the interface annunciations with the associated user–model transitions on the other. These two processes, or models, must “march” in synchronization and never encounter error states, restricting states, and 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 we are 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 up-shift manually when the machine was still in state **low-2**, the machine would transition to state **medium-1** (see Figure 1) and the interface 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**).

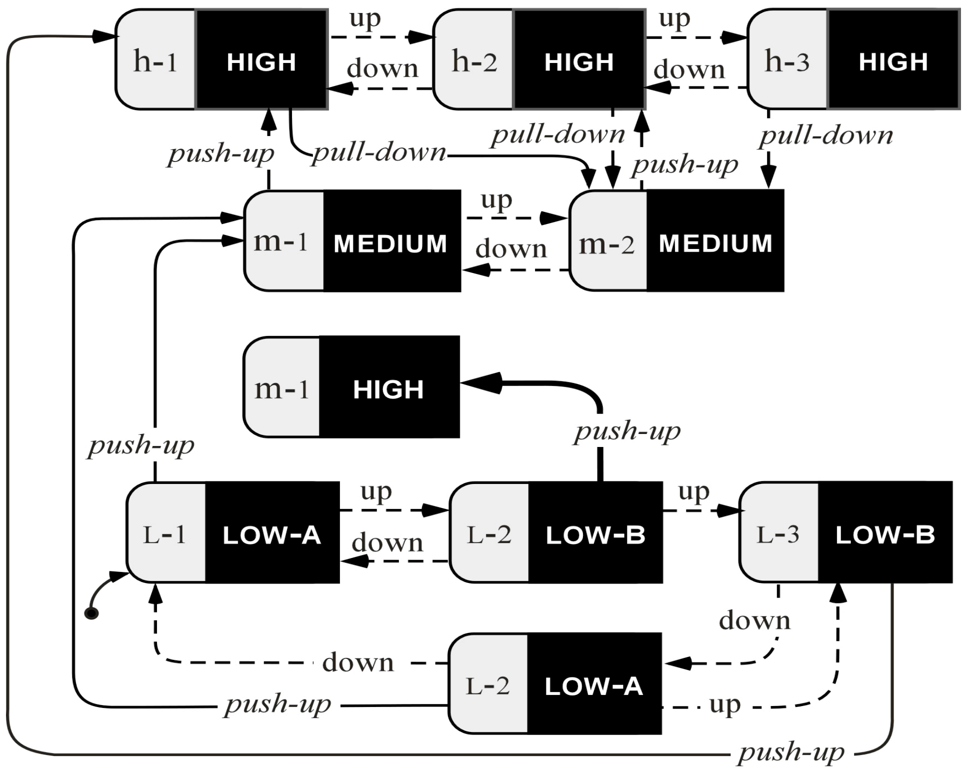


Figure 4. Composite of the machine and the user model

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, there is nothing we can do to alleviate the problem; no additional training, no better user manuals, procedures, or any other countermeasures will help. We 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, it is free of error states, restricting states, 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 & Degani (2002). Here we shall describe the underlying ideas and principles of the methodology and illustrate the procedure with the aid of examples, and in particular, the one that we are already familiar with—the transmission system.

Outline of the algorithmic approach

Our 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 1) they belong to the same specification class, 2) each user-triggered event that is available and active in one of the states is also available and active in the other, 3) whenever starting from either of the two states and triggered by the same event sequence, the state pairs visited, respectively, also satisfy conditions 1) and 2). 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 computed.

The next step of the algorithm consists of generating a reduced user model. The user model's states comprise 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 *minimal 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), and none of the selected maximal compatibles can be omitted from the selected set without violating the cover property (this constitutes the *minimality* property).

Once the state set of the reduced model has been selected as just 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 in this connection: (1) sets of distinct events that need not be distinguished and can be grouped together, (2) events that can be deleted since their presence in the reduced model is redundant, and (3) transition non-determinism 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. (The mathematical details of this algorithmic approach are discussed in Heymann & Degani (2002). Here they will only be demonstrated through the ensuing examples).

Compatible states

The user model must enable us to operate the machine correctly with respect to the user's tasks and requirements. Thus, while 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 us to distinguish between two internal states (say **medium-1** and **medium-2** of mode MEDIUM), if, following any event sequence, 1) we always end up in the same specification-class (e.g., HIGH), and 2) the same set of user-triggered events is available, regardless of which of the two internal states we 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 be 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 we identify and mark all incompatible pairs (i.e, those that cannot be grouped together on the interface), then, of course, the remaining state pairs must be compatible. We proceed as follows:

An immediate criterion for incompatibility of a state-pair is that the two states which 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 specification class and **high-1** belongs to HIGH. We must never group these two states together on the display.

The second criterion for designating a pair of states as incompatible has to do with event sequences. We mark a state pair incompatible if, starting from the two states and following the same sequence of events, we transition 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**. Since **medium-1** and **high-1** are already known to be incompatible (because they belong to two different specification class), 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 Paull & Ungar, 1959, and Kohavi, 1978) as described via our transmission example next. 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, we progressively detect all incompatible state pairs, thereby "resolving" the table; that is, we uncover all the state-pairs that are not found to be incompatible, and we designate them as compatible.

In the case of the transmission example, we have eight states and there are (n*[n-1]/2 = 8*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 sub-section regarding incompatible pairs, we use the following procedure 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), we mark their cell as incompatible.
2. For all other state-pairs, we write in their cells the next state-pair(s) 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**.

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

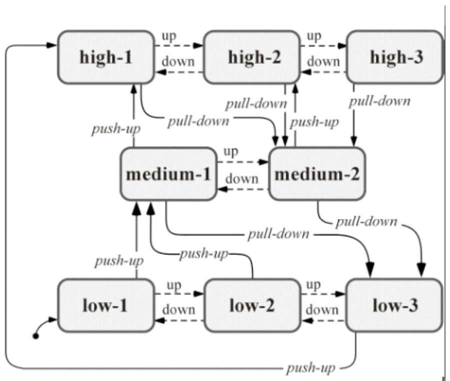


Figure 5. The merger table for the eight-state transmission system and its initial resolution

We begin at the top of the table. The upper-most cell represents the state pair **low-1** and **low-2**. Looking at the machine model (see the inset in Figure 5), and noting that states **low-1** and **low-2** transition on automatic up-shift [*up*], to **low-2** and **low-3**, we write that (**low-2**, **low-3**) inside the top cell. Next, we go down to the cell representing to the state pair **low-1** and **low-3**. We note that in the machine model, the states transition on manual up-shift (*push-up*) into **medium-1** and **high-1**, and that's what we write inside the cell. Moving one cell to the right, we now consider the cell representing **low-2** and **low-3**. Looking at the machine model, we note that there are two common transitions from this pair: an automatic down-shift (event *down*) from these two states takes us to **low-1** and **low-2**; and a manual up-shift (event *push-up*) takes us to **medium-1** and **high-1**. We write these two state-pairs in the cell.

Now, we go down the table to the cell representing **low-1** and **medium-1**. Since each state of this pair belongs to a different specification class, they are immediately deemed incompatible. The same applies for the **low-1** and **medium-2**. In this fashion we go cell by cell and populate the rest of the table. Notice, however, that the cell representing **high-1** and **high-3** is empty. This is because these two states are not incompatible (they both belong to HIGH and have push down as active user-triggered event), yet they don't transition into another state-pair under a common event like the rest of the state pairs. We therefore leave the cell empty, only to deal with it later.

Second iteration. We now continue with the resolution process. But from this step onward, we do not need to refer to the machine model anymore. In an iterative manner, we 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** and **low-2** contains the pair (**low-2**, **low-3**). We look into the cell representing **low-2** and **low-3** in Figure 5 and find in there two state-pairs: (**low-2**, **low-3**) and (**medium-1**, **high-1**). We write these two state-pairs inside the cell representing **low-1** and **low-2** (in Figure 6).

low-2	**[medium-1, high-1]** **[low-1, low-2]**						
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]**
	low-1	**low-2**	**low-3**	**medium-1**	**medium-2**	**high-1**	**high-2**

Figure 6. The second iterative resolution

Continuing with the procedure, we designate the cell representing **low-2** and **low-3** in Figure 6 as incompatible (because it contains the pair **medium-1** and **high-1**, which was already marked as incompatible). In the cell representing **medium-1** and **medium-2**, we place the state pair (**high-2**, **high-3**). The cell representing **high-1** and **high-2** gets the state pair (**high-1**, **high-2**), **high-2** and **high-3** get (**high-2**, **high-3**), while **high-1** and **high-3** stay empty as before.

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

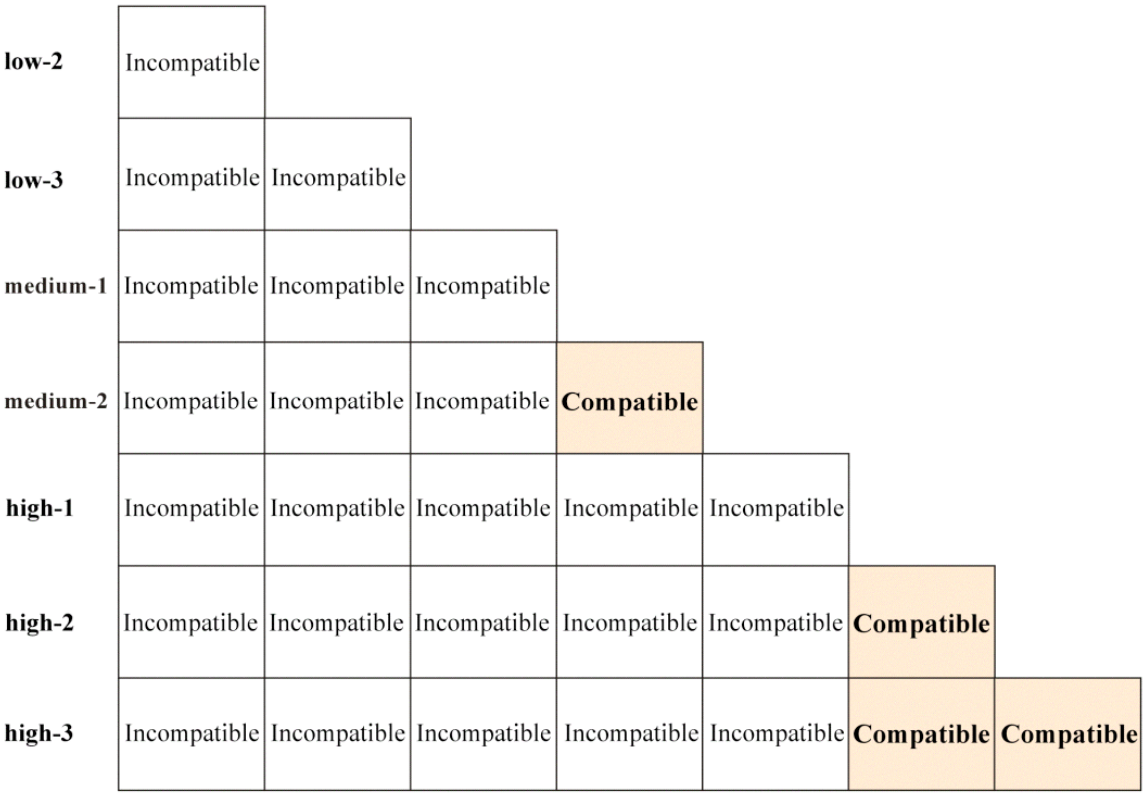
low-2	Incompatible						
low-3	Incompatible	Incompatible					
medium-1	Incompatible	Incompatible	Incompatible				
medium-2	Incompatible	Incompatible	Incompatible	$\begin{bmatrix}\textbf{high-2}\\ \textbf{high-3}\end{bmatrix}$			
high-1	Incompatible	Incompatible	Incompatible	Incompatible	Incompatible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

Figure 7. The third iterative resolution

Final iteration. In this step we realize that no additional incompatible pairs are identified, and the table remains identical to that of Figure 7. From here on, no further iterations will ever produce incompatible pairs. Therefore, we mark the empty cell of high-1 and high-3 as "compatible" (see Figure 8), concluding the resolution procedure.

The resolution procedure identified all the incompatible and compatible pairs. Figure 8 shows that we have four such compatible pairs: (**high-1**, **high-2**), (**high-1**, **high-3**), (**high-2**, **high-3**), and (**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.



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

Identifying compatible sets

Although we have identified all the compatible pairs in the system, and we know that we can combine the pairs so as to reduce the number of states (and corresponding display indications) to create an abstracted user model, we are not content yet. Why? Because it may be possible to further reduce the system by considering compatible triples, quadruples, etc. The idea here is based on the simple 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 our 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, we are interested in this reduction procedure to find what we call 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**) make up 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). But 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 above set of maximal compatibles forms the basis from which we now construct the user model. We combine **high-1**, **high-2**, and **high-3** into a single indication that we call HIGH; **medium-1** and **medium-2** into MEDIUM, and we provide separate indications for **low-1**, **low-2**, and **low-3**. The reduced user model obtained for the transmission system is shown in Figure 9. Note that both the MEDIUM and HIGH have self-loops. These are the internal events that take place "inside" MEDIUM and HIGH. But 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 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 information about these internal gear shifts in the user manual and/or perhaps provide some annunciation, e.g., blinking, about their occurrence, on the interface. It is noteworthy, however, that in the case of automotive transmission systems, most car manufacturers opt not to provide any indication to the user about internal gear shifts while the car is in automatic "DRIVE" mode).

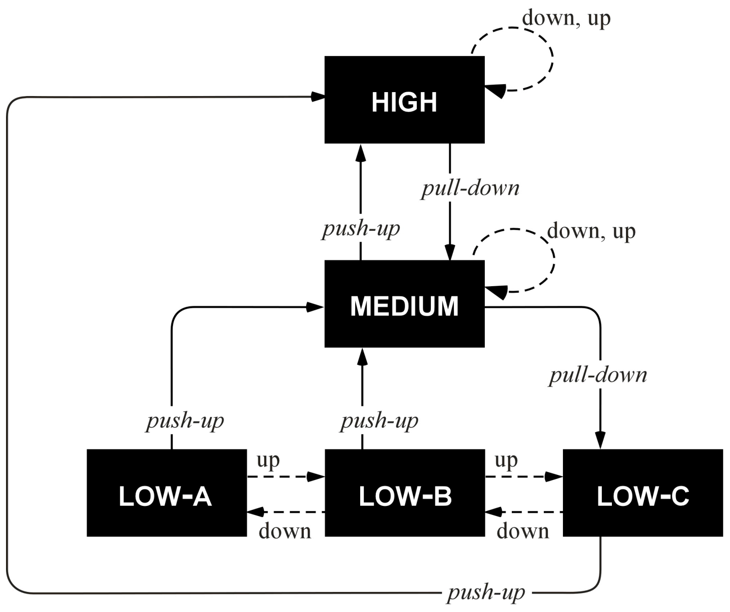


Figure 9. Succinct and correct user model for the transmission system

As a final check, we may wish to evaluate the user model by employing the verification procedure mentioned earlier in the paper, and making sure that no errors have crept in while constructing the interface or anywhere throughout the process (see Shiffman, Degani, and Heymann, 2005 for a computerized tool that can automatically verify large systems).

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. Along the same lines, we extract from the user model all the (event-related) information that must appear in the user manual and then specify the content. At this point we're done with the transmission system; we have reduced it as much as possible and come up with the specifications for a succinct and correct display and user manual information.

FURTHER ASPECTS OF THE REDUCTION PROCEDURE

The transmission system that we have 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, such as *ua* and *ud,* are user triggered; the rest are automatic). Four specification classes are defined for this machine: A, B, C, and D. The task requirement is similar to our 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.

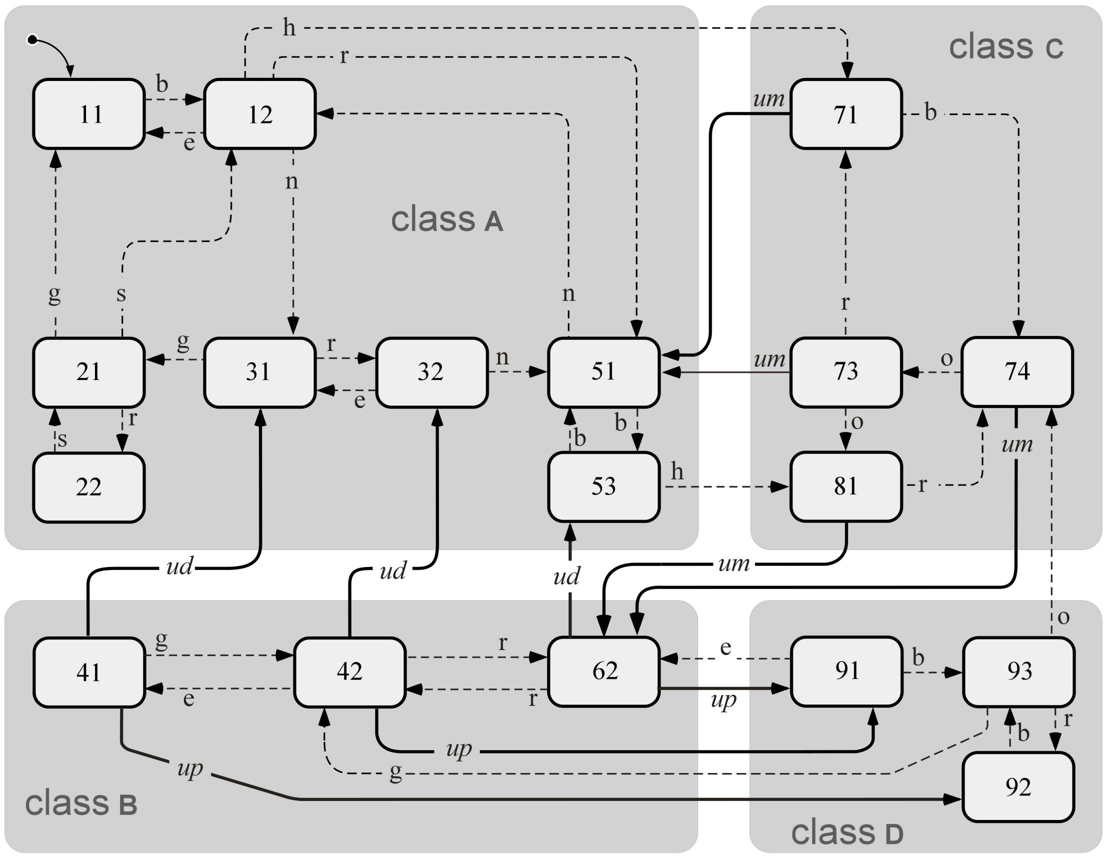


Figure 10. Machine model

We perform the reduction procedure 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 in the transmission system where 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 have the states **21, 22, 31**, and **32** which appear in all of them. 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 chose from: One consists of the compatibles **1, 4, 5, 6, 7,** and **8** as its state set, while the other consists of **2, 3, 5, 6, 7**, and **8**. (These two sets constitute the only possible minimal covers, as discussed earlier in the 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, etc. 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**, and **8** of the above list.

We now proceed to construct the user model. We first incorporate the selected maximal compatibles into user model states (modes A-1 and A-2; B; C-1 and C-2; and D). Next, we establish the transitions between the user model states in the following way: For each mode and each event label, we determine the set of all constituent machine model target states (to which an outgoing transition with this label exists). A transition with the corresponding label is then drawn from the mode under consideration to each mode that includes all the target states. This procedure results in the reduced model of Figure 11.

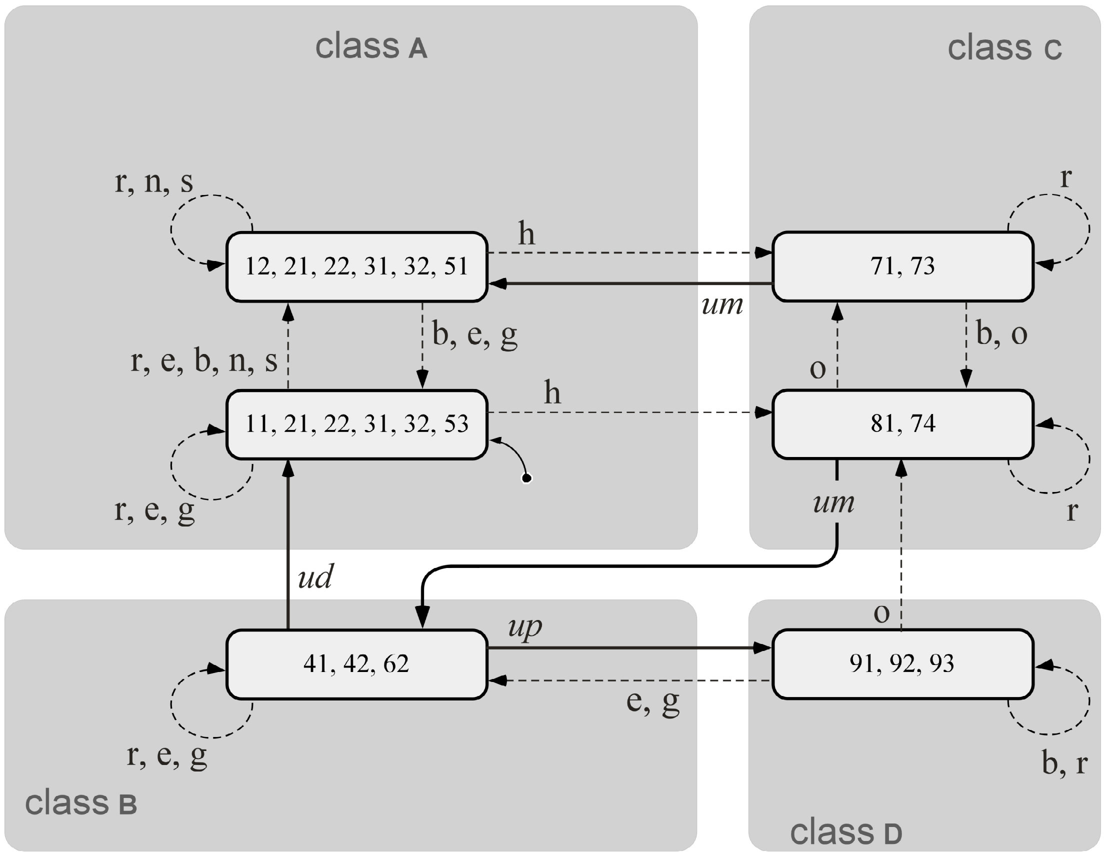


Figure 11. Reduced machine

Next, note that there may be non-deterministic outgoing transitions to states within the given specification class. (This non-determinism 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 to both A-1 and, as a self-loop, to A-2. We can eliminate this non-determinism by judicious decision as to which of the redundant transitions to delete. We further note 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 12. It contains only six modes.

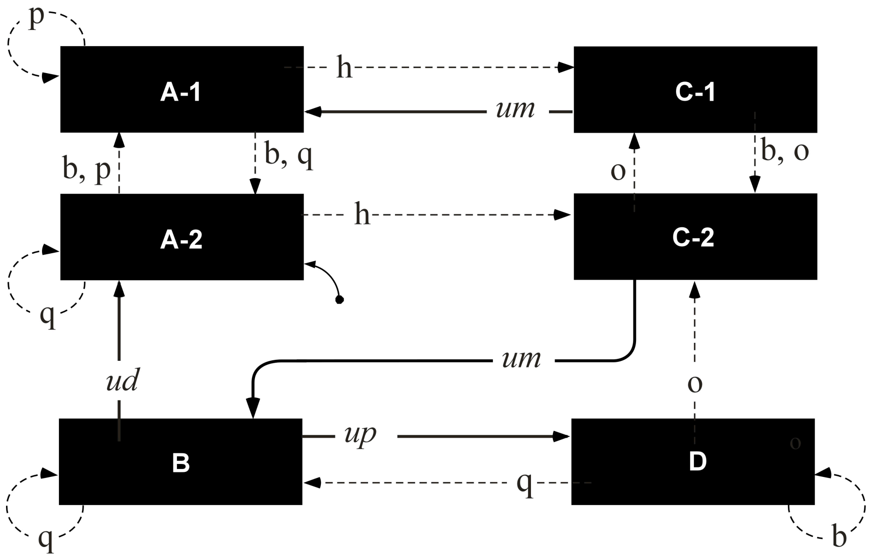


Figure 12. Correct and succinct user model and interface

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 either A-2 or to D, respectively. In C-1 and C-2 the user can trigger event *um*, leading the system to mode A-1 or B.

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: first and foremost was that the user model and interface were correct; second was that they were minimal, or succinct, in terms of the amount of information (e.g., mode annunciations, selection buttons, and parameter settings, as well as 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. While 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, there are many tools available today that allow designers to specify the system's behavior (see Harel and 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 and on the properties of the system that we need to insure. 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 (see, for example, 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, there are many algorithmic techniques to deal with this problem (e.g., Kam et al., 1997). Using a computerized tool (Shiffman, Heymann, & Degani, 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 our 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") so as to provide a smooth, effortless, and non-intermediary human-machine interaction. While 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's 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 may intuitively appear to be correct have been shown, after applying formal verification, to be faulty. While many interface and interaction problems are identified in simulations and usability testing, some are left unidentified and can plague a system for years. Furthermore, as systems become larger, more integrated (comprising several subsystems that are linked and synchronized), and therefore more complex, it is becoming more and 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. Due to the current intuitive and iterative approach for generating design solutions, there is never a guarantee that the selected interface solution cannot be further reduced. To this end, we believe 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, methodologies, procedures, and tools for generating correct interfaces, will help designers to better understand and reason about critical design issues that are currently addressed in an intuitive, ad hoc, way.

ACKNOWLEDGMENTS

This work was conducted as part of NASA's base research and technology effort on human-automation interaction. 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 encouragements 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*. Presses Universitaires de Namur.

Bosser, T., & Melchoir, E. M. (1990). The SANE toolkit for user centered design, prototyping, and cognitive modelling. *Proceedings of the Annual ESPRIT Conference*. Brussels, Belgium: London, U.K: Kluwer Academic Publishers.

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*. Springer-Verlag.

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

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

Degani, A., Heymann, M., & Shafto, M. (1999). Formal aspects of procedures: The problem of sequential correctness. *Proceedings of the 43rd Annual Meeting of the Human Factors and Ergonomics Society*. Houston, TX: Human Factors Society.

Degani, A., & Heymann, M. (2002). Formal Verification of Human-Automation Interaction. *Human Factors, 44*(2), 28-43.

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.

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*(2): pp. 107-131.

Federal Aviation Regulation 25.1329. *Certification of Flight Guidance Systems.* Washington, DC: Code of Federal Regulations.

Foley, J. D., and Wallace, V. L. (1974). The art of natural graphic man-machine conversation. *Proceedings of the IEEE, 62*(4), 462-471.

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

Harel D., and 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 Technical 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(4), 259-264.

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

Kam, T., Villa, T., Brayton, R., & Sangiovanni-Vincentelli, A. (1997). Implicit Computation of Compatible Sets for State Minimization of ISFSMs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16* (6), 657-676

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

Krzystrof, G, & Weld, D. (2004). SUPPLE: Automatically generating user interfaces. *Proceedings of the 2004 International Conference on Intelligent user interfaces* (IUI-04). Madeira, Spain January 13-16.

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

Lecerof, A., & Paternò F., (1998). Automatic Support for Usability Evaluation. *IEEE Transactions on Software Engineering*, 24(10), 863-888.

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: National Technical Information Services.

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

Norman, D. (1983). Design rules based on analysis of human error. *Communications of the ACM, 26*(4), 254-258.

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

Palanque, P., & Paternò, F. (1998). *Formal Methods in Human Computer Interaction.* Springer-Verlag.

Parasuraman, R., Sheridan, T.B., & Wickens, C.D. (2000). A model for the types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 30*(3), 286-297.

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* (pp. 379-385).

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(2), 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*, 356-367.

Rodriguez, M., Zimmerman, M., Katahira, M., de Villepin, M., Ingram, B., & Leveson, N. (2000). Identifying Mode Confusion Potential in Software Design, *Digital Aviation Systems Conference*.

Rushby, J. (1999). Using Model Checking to Help Discover Mode Confusions and Other Automation Surprises, *3rd Workshop on Human Error, Safety, and System Development*, Liege, Belgium, 7-8 June.

Rushby, J., (2001). Analyzing Cockpit Interfaces using Formal Methods. *Electronic Notes in Theoretical Computer Science*, Volume 43, http://www.elsevier.nl/locate/entcs/volume43.html

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, E., & Slacher, E. (1995). Declarative interface models for user interface construction tools: The MASTERMIND approach. In *Proceedings of the European Human Computer Interaction Conference* (Grand Targhee Resort, August).

Shiffman, S., Degani, A., & Heymann, M. (in press). Uiverify - a web-based tool for verification and automatic generation of user interfaces. *Proceedings of the 8th Annual Applied Ergonomics Conference*, March 21-24, 2005, New Orleans.

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*. Springer-Verlag, 281-301.

Wasserman, A. I. (1985). Extending state transition diagrams for the specification of human-computer interaction. *IEEE transactions on Software Engineering, SE-11*(8), 699-713.

MICHAEL HEYMANN received the B.Sc. and M.Sc. degrees from the Technion, Haifa, Israel, in 1960 and 1962, respectively, and the Ph.D. degree from the University of Oklahoma, Norman, in 1965, all in Chemical Engineering. For the past 33 years he has been with the Technion, Israel Institute of Technology, where he is currently Professor of Computer Science and Director of the Center for Intelligent Systems, holding the Carl Fechheimer Chair in Electrical Engineering. He has previously been with the department of Applied Mathematics, of which he was Chairman, and with the Department of Electrical Engineering. Since 1983 he has also been associated with NASA Ames Research Center.

His past research has covered topics in the areas of linear system theory, differential games, optimization and adaptive control. His current interests are primarily in the areas of discrete event systems, hybrid systems, the theory of concurrent processes, and various formal aspects of human-machine interaction. He has been on the editorial boards of the *SIAM Journal of Control and Optimization* and *Systems and Control Letters*.

ASAF DEGANI received the Practical Engineer degree in Architecture from ORT Technical College, Israel, and the B.Sc. from Florida International University. He received the M.Sc. in Ergonomics from University of Miami and the Ph.D. in Industrial and Systems Engineering from Georgia Institute of Technology, Atlanta. Since 1989 he has worked as a researcher at the Human Factors and Computational Science Divisions at NASA Ames. His research interests are in the areas of human interaction with automation, modeling and formal methods, procedure design, and aviation safety.