

On abstractions and simplifications in the design of human-automation interfaces

Michael Heymann

Department of Computer Science
Technion, Israel Institute of Technology
heymann@cs.technion.ac.il

Asaf Degani

NASA Ames Research Center, California
adegani@mail.arc.nasa.gov

ABSTRACT

This report addresses the design of human-automation interaction from a formal perspective that focuses on the information content of the interface, rather than the design of the graphical user interface. It also addresses the issue of the information provided to the user (e.g., user-manuals, training material, and all other resources). In this report, we propose a formal procedure for generating interfaces and user-manuals. The procedure is guided by two criteria: First, the interface must be correct, i.e., that with the given interface the user will be able to perform the specified tasks correctly. Second, the interface should be as succinct as possible. The report discusses the underlying concepts and the formal methods for this approach. Several examples are used to illustrate the procedure. The algorithm for constructing interfaces can be automated, and a preliminary software system for its implementation has been developed.

INTRODUCTION

Human interaction with automation is so widespread that almost every aspect of our lives involves computer systems, information systems, machines, and devices. These machines are complex and are comprised of many states, events, parameters and protocols. Yet, the only *face* the user sees is the interface: always a (highly) reduced description of the underlying behavior of the machine. This is no coincidence, because otherwise the user would be subjected to enormous unnecessary complexity. Consider for example, consumer electronics where making the user-interfaces and associated user-manuals as efficient, simple, and succinct as possible is becoming a marketing imperative, and no longer is just an engineering and human factors ideal. As consumer devices get increasingly complex and multifunctional, there is a reciprocal drive to render them simpler and easier to use (and thereby more marketable).

In the majority of today's automated systems, the human is the supervisor. Users interact with systems or tools to achieve certain operational tasks (Parssuramann et al., 2000). These tasks, or task specifications, may involve the execution of specific sequences of actions (e.g., a procedure for setting up a medical radiation machine), monitoring a machine's mode changes (e.g., an automatic landing of an aircraft), or preventing a machine from reaching specified illegal states (e.g., tripping a power grid). To achieve these task specifications, the user is provided with information about the behavior of the machine. In most cases, this information is provided by means of an interface and associated user-manuals and other training material.

Naturally, for the user to be able to interact with the machine correctly and reliably so as to achieve the task specification, the information provided to the user must first and foremost be correct. For example, if the pilot of an airliner has insufficient information to resolve a mode transition and to decide whether, after entering a command to the autopilot, the aircraft will enter "climb" mode or "level-flight" mode, then one can say that the information provided to the pilot is

inadequate. One sure way to guarantee sufficient information for correct interaction is to provide the user with *the full detail* of the machine behavior. This way the user can, in principle, always track the status of the machine correctly and reliably. But this amount of detail has an obvious downside too; the size of interfaces and weight of user manuals will be huge, and the burden on the user incomprehensible and unmanageable.

In practice, the interface and related user manuals are always a reduced, or abstracted, description of the machine's behavior. No interface provides a complete description of the underlying behavior of the machine. Therefore, a major concern of designers of automated systems is to make sure that these abstracted interfaces and manuals are indeed adequate and correct. Currently, this evaluation is performed in an *ad hoc* fashion. It usually involves costly simulations and extensive testing, and in industries such as aerospace and medical equipment, it also involves complicated certification procedures (see for example Federal Aviation Regulation 25.1329 and associated Advisory Circular). Yet, despite the best efforts by design teams and certification officials, numerous incidents and accidents involving incorrect interfaces have been reported in aviation (Abbott, Slotte, and Stimson, 1996), maritime (National Transportation Safety Board, 1997), medical (Leveson, 1995 see Appendix A -- the Therac-25 accidents), and automotive systems (Andre and Degani, 1997). Even in simpler consumer devices, flaws in the user interface design are frequently encountered.

Developing a correct interface is only one requirement. In addition, we all strive for interfaces and user-manuals that are simple and easy to use. One basic aspect of this requirement is to develop interfaces and user-manuals that are succinct. That is, the number of states and events that the user needs to understand and track in order to operate the system correctly should be as small as possible. Currently, the design decisions as to what information must be provided to the user, both

in the interface and in user-manuals, are made intuitively. Systematic methodologies do not exist for these decisions.

One of the outcomes of having incorrect and extremely complex interfaces is a common problem called “automation surprises,” where operators (e.g., pilots, technicians, users) have difficulty understanding the current status of an automatic system as well as the consequences of their interaction with it (Woods, Sarter, and Billings, 1997).

In an earlier paper and a recent NASA technical memorandum (Degani and Heymann, in press: 2000), we discussed a methodology for evaluating interfaces and user manuals. Given a description of the machine, specifications of the user’s task, interface, and all relevant information the user has about the machine, the procedure evaluates whether the interface and user manual information are correct for the task. That is, can the user achieve all the specified tasks correctly and reliably, given all the information provided? The proposed procedure can be automated and applied to the verification of large and complex human-machine systems.

In the present report we take an additional step and discuss a formal methodology for automatic generation of interfaces and user manuals. The requirement, of course, is that the interfaces and user manuals be both correct and succinct. The design problem can be formulated as follows: The machine and the user’s operational requirements (task specifications) are given. Now the problem is to generate an interface and associated user information that enables the user to interact with the machine *correctly*. It is further required that the interface and all user information be as simple and as succinct as possible. Naturally, additional considerations must be taken into account to ensure efficient human-machine interaction. These include graphical user interface design, cognitive limitations, human physical abilities, and the like. But underlying all is the basic correctness issue on which we focus our attention here.

The report is organized as follows: We begin by discussing the four components of human-machine interaction that are part of our theory and methodology: the machine, the task specification, the interface, and user model. We then use these four elements to verify the correctness of a proposed interface for a given machine. Next, we turn to the main topic of this report, a formal methodology for constructing interfaces and related user information (e.g., user-manuals). Here we describe a procedure for abstracting a machine model to the most succinct description that enables correct user-machine interaction. We illustrate this procedure with an example of a transmission system in a car and then show other characteristics of abstraction using an example of a somewhat more complex machine. Finally, we conclude with a brief summary and discuss some of the implications of this work for designers of automated systems.

FORMAL ASPECTS OF HUMAN-AUTOMATION INTERACTION

Many aspects of the human-machine interaction, such as the design of interfaces in terms of their graphical appearance (which is still highly empirical and intuitive), are not amenable to formal analysis and design. Yet aspects of interaction that concern the information content provided to the user about behavior of a system can be formally analyzed, and thus can be systematically verified and designed. Here the emphasis is on questions regarding “what” information must be provided to the user and “when,” rather than on “how” this information is to be presented.

In this work we focus primarily on the information content provided to the user about the behavior of a system. This aspect of user interaction with machines can be described and analyzed formally by considering the following four elements: (1) the machine-model, (2) the operational tasks, (3) the machine’s interface with the user, and (4) the user’s model of the machine, i.e., the information provided to the user about the machine behavior (e.g., in the user manual). Let us briefly review these elements.

Machine

As stated earlier, we consider machines that interact with their environment and specifically with their human users. We focus our attention on the behavior of machine states, transitions, and events. The machines are modeled as state transition systems (in particular finite state machines). A *state* represents a mode, or configuration, of the machine. Transitions represent discrete-state (mode) changes that occur in response to events that trigger them. Some of the transitions occur only if the user triggers them, while other transitions occur automatically and are triggered by the machine's *internal* dynamics, or its *external* environment.

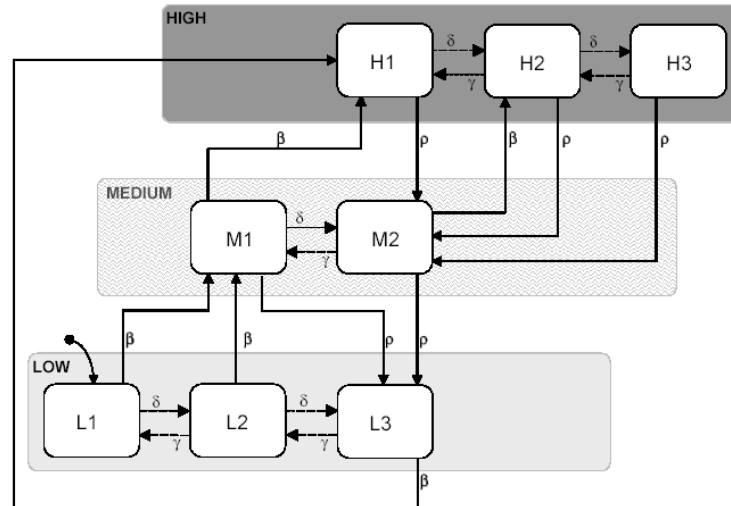


Figure 1. Transmission system.

To illustrate a typical machine model, let us consider the machine of Figure 1, which describes a simplified multi-mode three-speed transmission system proposed for a certain vehicle. We use the convention that user-triggered transitions are described by solid arrows, while automatic transitions are depicted by dashed arrows. The transitions are *labeled* by symbols to indicate the (triggering) circumstances under which the machine moves from state to state. The transmission has eight states, or modes. These modes are grouped into three super-modes that represent manually switchable gears (or speeds): *low*, *medium* and *high*. The states within each speed represent internal torque-

level modes. Thus there are torque modes $L1, L2, L3$, in the low speed super mode; there are torque modes $M1, M2$, in the medium speed super mode; and modes $H1, H2, H3$, in the high speed super mode. The transmission shifts automatically between torque modes (based on torque, throttle, and engine and road speeds). The automatic up-shifts (to higher torque modes) are denoted by the event symbol δ and the automatic down-shifts by the symbol γ . The (user operated) manual speed changes, achieved by pushing a lever up or down, are denoted in the Figure by the event symbols β and ρ , respectively. Pushing the lever up shifts to a higher speed and pushing down shifts to a lower speed. The transmission is initialized in the low torque mode $L1$ of the low speed (as indicated in the Figure by the free incoming arrow).

Task Specifications

The second element is the specification of the operational tasks the user is required to perform while using the machine. For example, a common task specification in an automated control system is that the user be able to determine unambiguously the current and the subsequent mode of the machine.

In terms of a formal description, the task specification consists of a partition of the machine's state-set into disjoint clusters that we shall call specification classes (or modes) that the user is required to track unambiguously. In other words, does the user know whether the system is currently in, or is about to enter into, the super-mode High, Medium, or Low? We note that the user is not required to track every internal state change of the machine: for example, between the modes $L1, L2$ and $L3$ inside mode Low.

Interface

The third element is the user interface. In practice, the interface consists of a control unit through which the user enters commands (e.g., mode selections, parameter changes) into the

machine, as well as a display through which the machine presents information to the user.

Generally, the interface provides the user a simplified view of the machine. Not all the events of the machine are announced to the user, and the interface displays only partial information about the actual behavior of the machine.

Formally, the interface consists of a listing and description of the events accessible to the user. These include, of course, *all* the user-triggered events (inputs to the machine), but generally only a subset of the events that are associated with automatic transitions. This is because some of the latter are not monitored at all, and others are monitored only in groups. The interface announcement tells the user only that one of the events in the group took place, without specifying which.

It is noteworthy that events *per se* cannot be displayed in the interface. What can be displayed is some consequence of their occurrence. Therefore, events are usually represented by *display modes* that become active as a result of the event occurrence. (How these modes are presented to the user graphically (e.g., icon shape, color, etc.) is beyond the scope of this report).

To illustrate, let's return to the multi-mode transmission model of Figure 1. The system in Figure 2 gives one possible user interface for this model. Here the monitored events are only the ones triggered by the user. In the Figure 2 we have also provided a description of the three display modes, as well as how the user would observe the machine's behavior when all automatic transitions are internalized and unobserved. Note that the torque modes are completely suppressed from view.

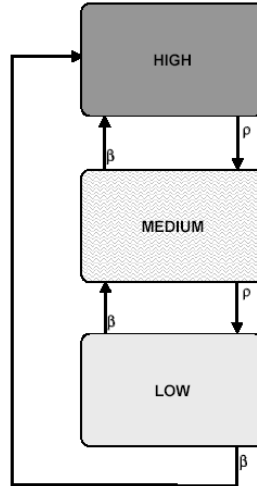


Figure 2. Proposed interface and user model.

An alternate interface for the transmission is provided in Figure 3. Here the monitored events consist of the user-triggered events as well as the automatic transitions. Again, we provide a possible description of how the user might observe the machine behavior. Note that wherever the automatic transitions do not trigger a state change in the user model, they are shown by (gray) self-loops to indicate the fact that the user-model “is aware” of the possibility that these events might take place without its actual participation.

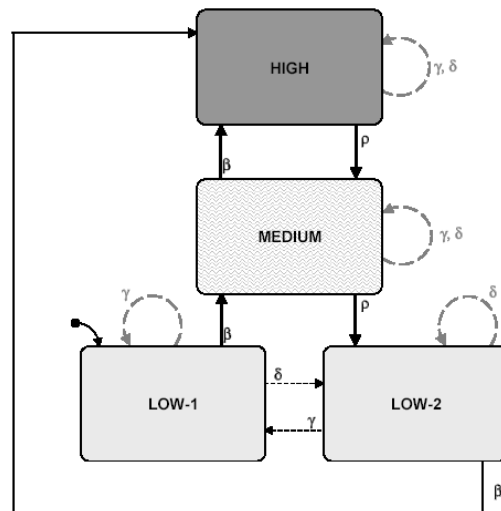


Figure 3. Alternate interface and user model.

User model.

As mentioned earlier, the interface provides the user with a simplified view of the machine, in that it displays only partially the machine's internal behavior. The description of the machine's operation that is provided to the user is generally also an abstracted simplification of the actual machine behavior. This description is usually provided in terms of a user manual, training material, formal instruction, or any other means of teaching the user; however, it is presented here as a formal model that we refer to as the *user model* of the machine. By its very nature, the user-model is based on the interface through which the user interacts with the machine, and thus relates to the modes and events that are displayed there. Therefore, for analysis purposes the interface events and modes are all explicitly referred to in the user-model, and in this respect can be thought of as “embedded” in the user-model.

Let us return to the user interface displayed in Figure 2. This Figure depicts a possible user-model associated with the interface that monitors only the user-triggered events for the transmission system. This particular user-model can be obtained from the machine model of Figure 1 by suppressing (internalizing) the events that are not monitored, and grouping the states as suggested by the specification. It can be seen that the manual shifts from *MEDIUM* up to *HIGH* or down to *LOW*, as well as the down-shift from *HIGH* to *MEDIUM*, are always completely predictable. However, the up-shift from the *LOW* gear depends on the current torque mode. Note that the up-shifts from L1 and L2 switch the transmission to *MEDIUM* speed, while the up-shift from L3 switches the transmission to the *HIGH* speed. Therefore, from the suggested interface of Figure 2, it cannot be predicted whether the up-shift will lead the transmission from *LOW* to *MEDIUM*, or to *HIGH* gear.

An alternate user-model for the transmission model is presented in Figure 3. This user-model describes an interface that also monitors the occurrences of two specific automatic transitions, in addition to all user-actuated events. This user-model, in particular, is aimed at enabling the operator

to determine whether the transmission is in a display-mode *LOW-1* (where an up-shift is supposed to lead to *MEDIUM* speed), or in the display-mode *LOW-2* (where an up-shift leads to *HIGH*).

Correctness of interaction

Among the four elements that play a role in the human automation interaction, the machine model and the task specification must be regarded for our purpose as given and beyond dispute, because they are not subject to our scrutiny. In contrast, the interface and the user model, which are the subject of investigation in the present report, must be examined for correctness. Specifically, we wish to know whether a given interface and user model enable the user to operate the machine correctly so as to satisfy the specification.

This verification problem was the focus of a recent paper (Degani and Heymann, in press) in which a methodology was described for verification of user-model and interface correctness for a given machine-model and specification. It was shown that the user model and interface are correct if, in a *composite model* obtained through a suitably defined *synchronous composition* of the machine model and the user model (see Figure 4), there exist no *error states* and no *blocking states*. An error state represents a divergence between the machine and user models – the user model does not indicate the correct specification mode the machine is in! A blocking state is one in which the machine can trigger a monitored transition that the user-model does not recognize.

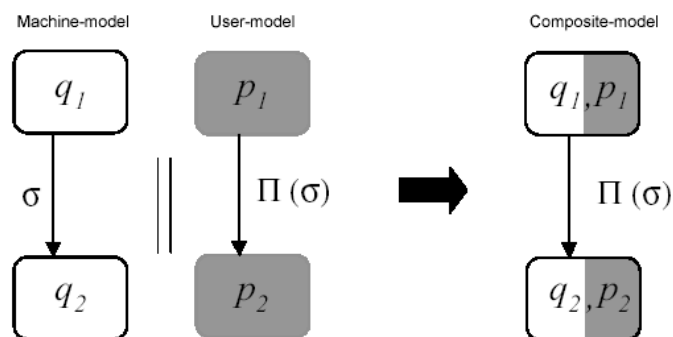


Figure 4. Masked synchronous composition

Next, we briefly review the discussion of our recent paper (Degani and Heymann, in press) about the verification problem. This will also introduce us to terminology that will be required for the discussion of the main issues of the present report.

As we have already stated, the interface and user-model are intended to provide an abstracted and reduced description of the machine. This abstracted description does not enable the user to determine with certainty each state the machine is in, since it is required only that the user be able to determine which specification-class (mode) the machine is in and which it is about to enter. Let Σ_M denote the set of events, or *transition labels*, that take place in the actual machine model. The events that ultimately appear in the associated user-model and are displayed in the interface constitute a reduced subset of the set Σ_M of machine events. This reduction, or abstraction, is achieved through a projection operation $\Pi : \Sigma_M \rightarrow \Sigma_{USR}$ as explained next, where Σ_{USR} is the event set that is displayed in the interface and appears in the user-model.

The event set Σ_M consists of three disjoint subsets: (1) Σ_M^o - the set of *observed-events* that includes all machine events that are actually presented in the interface and appear also in the user-model; (2) Σ_M^m - the set of *masked events* (that are not displayed individually, but rather are grouped into sets of two or more events each, with each set having a single event-label in the user-model); and (3) Σ_M^u - the set of *unobserved-events* that are neither displayed nor appear in the user-model.

In view of the above, the event set Σ_{USR} of the machine's user-model consists of the union of the event sets $\Pi(\Sigma_M^o)$ (which is identical to Σ_M^o), the event set $\Pi(\Sigma_M^m)$ which denotes the set of events obtained after masking the events in Σ_M^m , and the "empty event" ε ($= \Pi(\Sigma_M^u)$) that represents the set of unobserved events.

In actual operation, the machine is driven by events from Σ_M . The user tracks the progress of the machine via the interface (display), where he or she observes events in Σ_{USR} , with the aid of the associated user-model. Thus, the user-model and the machine evolve concurrently. But they are only partially synchronized, in that the user-model tracks the actual state evolution of the machine with some uncertainty. This is because (1) not all machine events are observed and some machine-events are masked, and (2) the user-model is only an abstraction of the actual machine's behavior.

Suppose that the machine is at state q at which a transition labeled α is defined, leading to a state q' (we denote this by $q \xrightarrow{\alpha} q'$). Assume that when the machine is at state q , the user-model is at a corresponding state p . Event α can be either *observed*, *masked*, or *unobserved*.

If α is an observed event and hence $\Pi(\alpha) = \alpha$, it is required for adequacy of the user-model that a corresponding transition be also defined at configuration p , leading to p' . That is, there must exist a transition $p \xrightarrow{\alpha} p'$. In the concurrent operation of the machine and the associated user-model, there will appear a transition labeled α from the state pair (q, p) to the state pair (q', p') . That is, there will be a “composite” transition $(q, p) \xrightarrow{\alpha} (q', p')$.

If α is a masked event, there will be a corresponding transition $p \xrightarrow{\Pi(\alpha)} p'$ in the user-model, where $\Pi(\alpha)$ is the (masked) image of α in Σ_{USR} . The composite transition will appear as $(q, p) \xrightarrow{\Pi(\alpha)} (q', p')$. The fact that the event labels are taken from the user-model is because the composite transition is viewed from the point of view of the user.

Finally, if α is unobserved and $\Pi(\alpha) = \varepsilon$, the composite transition will appear as $(q, p) \xrightarrow{\varepsilon} (q', p)$, since there is no corresponding transition in the user-model and the transition is “viewed” by the user as the empty or *silent* transition.

For the user-model to be correct for the task specification, it is necessary that the user-model be able to track the machine-model's specification classes unambiguously. More explicitly, it is required that when the user-model enters a state p in response to an observed event string t , all possible states q that the machine-model could have entered in response to machine event strings s for which $\Pi(s) = t$, would belong to the same specification class.

Before proceeding with the discussion, let us use our methodology to verify whether the user model of Figure 3 is correct. Recall that this user-model is aimed at enabling the operator to determine unambiguously which speed the transmission is in or is about to enter. The composite model of the machine of Figure 1 and the user model of Figure 3 is shown in Figure 5. Here we can readily see the error state ($M1, High$) which is entered upon executing the event sequence $\delta\beta$ (δ followed by β). It is evident that the user model of Figure 3 is incorrect.

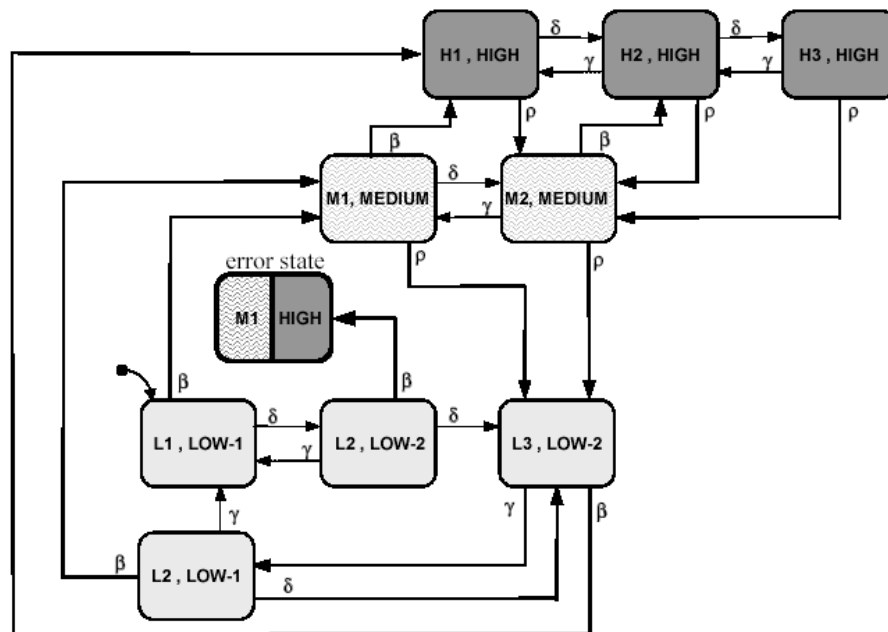


Figure 5. Composite model of the alternative user model.

It is of course possible to try out other interfaces and user-models and then employ the verification procedure to determine their correctness. However, such an approach is not likely to be very fruitful: It may take considerable effort to develop and verify one design after the other, with no guarantee of success. Furthermore, even when a correct interface is found, there is no assurance that it is the simplest. The development of a systematic approach for constructing interfaces that are *both* correct and succinct is the subject of the next Section.

MACHINE MODEL REDUCTION

In the previous section we have seen which conditions the user-model and interface must satisfy in order to enable the user to perform correctly a specified task on a given machine. We have also reviewed a procedure for verifying that these conditions indeed hold true. However, the question remains open as to how a correct interface and user model can be designed systematically for a given task.

As mentioned earlier, one possible choice of user model is to take the full machine model as user model and the complete machine event set as the set of monitored events. If the machine model is deterministic (as we assume throughout this report), this will insure that there will never be any problem in predicting the next state of the machine. But the operator would be required to track every state and every event in the machine – a formidable and impractical job. In the simple example of Figure 1, the machine has 8 states, 18 transitions and 4 distinct transition labels. But this is a tiny number when compared to “industrial size” situations.

In this section we shall turn to the main issue of the present and describe a procedure for the generation of *all* optimal user models and interfaces for a given machine model and task specification. In particular, we shall consider the problem of constructing, for a given machine and task specification, the set of all *best* possible user-models and event abstractions that satisfy the

specification. Here, by *best* user models and interfaces we mean the ones that cannot be further reduced! Since, as we shall see, these user models (and associated event abstractions) are generally not unique, we cannot speak of user-model “synthesis,” but rather, of machine model *reduction*. We shall show how all “smallest” user models and associated interfaces can be derived.

Compatible state sets and covers

We assume that the machine-model is given as a state machine and that the task specification is given as a partition of the state-set into disjoint classes of states that we refer to as *specification classes* (Degani and Heymann 2001). Thus, each state of the machine model belongs to a unique specification class. (In Figure 1 which depicts the multi-mode three speed transmission, the specification classes consist of the three speeds; Low, Medium and High. Each state, or mode, belongs to exactly one speed.)

Let us consider a machine-model given as a state-machine, and let the task specification consist of a partition of the machine-model’s state set Q into disjoint specification classes Q_1, \dots, Q_l (as described, for example, in Figure 1 where $l = 3$).

The user model must enable the user to operate the system correctly with respect to the specification classes. That is, it must enable the user to track the specification classes but not necessarily individual states. Thus, the user does not need to be able to distinguish (by means of the user model and interface) between two states p and q of the same specification class, if for the purpose of tracking the specification classes unambiguously it is sufficient for the user to know that the machine visited *either* p or q . More explicitly, the user does not need to be able to distinguish between p and q if the specification class visited following any user-machine interaction starting in state p , is the same as the specification class visited following the same user-machine interaction starting at state q . This leads to the following definition: Two states, p and q , are *specification*

equivalent (or *compatible*), if given that the machine is presently in either state p or q (of the same specification class), the specification classes to be visited under future inputs will be the same.

Stated more formally, we have

Definition: Two states p and q are specification compatible if and only if the

following two conditions both hold:

1. The states p and q belong to the same specification class,
2. If p' and q' are states such that there exists an event string $s = \sigma_1 \dots \sigma_n$ for which $p \xrightarrow{s} p'$ and $q \xrightarrow{s} q'$ are both defined, then p' and q' belong to the same specification class.

It is clear that if the only concern is to track the specification classes, two specification compatible states need not be distinguished in the user model. We may also conclude immediately that any set of states is specification compatible if all the pairs of states within that set are specification compatible.

Thus, if an efficient procedure is found for computation of all specification compatible pairs, the set of all compatible state sets will easily be computed. Indeed, the compatible triples will be obtained as the state triples, all of whose pairs are compatible; compatible quadruples as the quadruples all of whose triples are compatible, and so on.

Next, we have the following:

Definition: A set C of compatible sets of states is called a *cover* of the state set of the machine-model, if every state of the machine-model is contained in one or more elements of C .

Since a set that consists of a single state is (trivially) compatible, it follows that every state is included in at least one compatible set, so that the set of all compatibles is always a cover.

Definition: A compatible set of states is called a *maximal* compatible set, if it is not a proper subset of another compatible set; that is, if it is not contained in a bigger compatible set of states.

Since sets that consist of a single state are compatible, it is clear that every state is contained in at least one maximal compatible set. It follows that the set of maximal compatibles is a cover.

Definition: A cover C of compatibles is called a *minimal* cover, if no proper subset of C is a cover.

Of particular interest to us will be the set of all minimal covers formed from the set of maximal compatibles. That is, we shall be interested in minimal covers whose component elements are maximal compatible sets. In general, the number of such minimal covers can be greater than one.

We shall see below that minimal covers by maximal compatibles constitute the foundation of the model reduction and interface generation procedure. However, we shall first show the set of compatibles is computed.

Generation of compatible pairs

As stated above, the computation of compatible sets hinges on the construction of the set of all compatible pairs. An efficient iterative algorithm for construction of compatible state pairs is based on the use of merger tables (see e.g., Paull and Ungar 1959, and Kohavi 1978, where related model reduction problems are discussed).

L2							
L3							
M1							
M2							
H1							
H2							
H3							
	L1	L2	L3	M1	M2	H1	H2

Figure 6. Table of all pairs

A merger table is a table of cells representing distinct state pairs. An initial table for the eight states of our transmission example is shown in Figure 6. Each cell of the table corresponds to a pair of distinct states, and each pair of distinct states appears in the table exactly once.

Next, we have the following observations that can be easily derived from the definition of compatible pairs:

A state pair (p, q) of the same specification class is **compatible** if and only if for every event symbol σ such that $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined, it is true that either $p' = q'$, or the pair (p', q') is compatible.

We shall use the above characterization of compatible sets to obtain a complementary characterization of all pairs that are **not** compatible (or incompatible). It will then be convenient for us to compute recursively the set of all incompatible pairs. The set of compatible pairs will then consist of all state pairs that are not found to be incompatible. Based on the above characterization of compatible pairs, the characterization of incompatible pairs is as follows:

A state pair (p, q) is **incompatible** if and only if either p and q belong to distinct specification classes, or there exists an event symbol σ for which $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined, and the state pair (p', q') is incompatible.

Using the above observations regarding compatible and incompatible pairs, the determination as to whether a state pair is compatible or incompatible is computed iteratively as follows.

1. For each state pair (p, q) that can be determined as incompatible in the first step based on the above characterization (i.e., if p and q belong to distinct specification classes), we mark the corresponding cell F (for false). For all other state pairs, we write in their cells their associated *transition pairs* that consist of all distinct state pairs (p', q') for which there exists an event symbol σ , such that the transitions $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined.

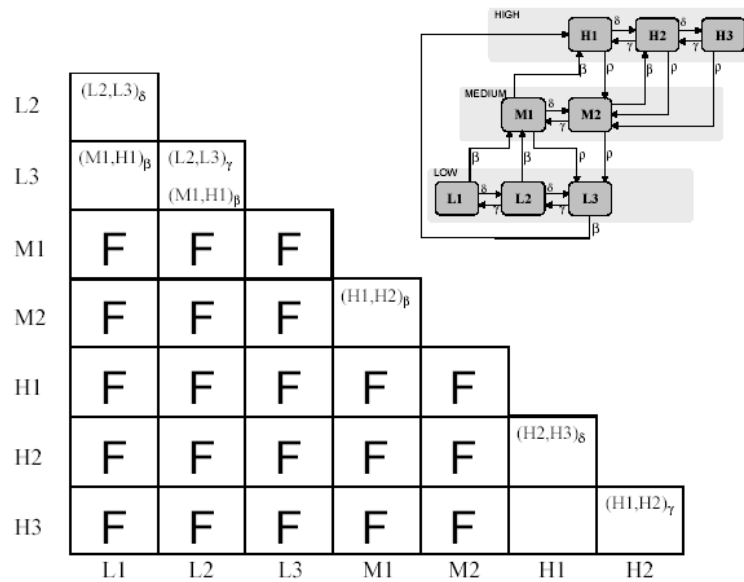


Figure 7. Resolution table (initial).

For illustration, the initial resolution table for the transmission model of Figure 1 is presented in Figure 7. Notice that each transition pair in the table has been subscripted with the associated event label. This subscription is not essential to the algorithm and is for the reader's convenience only. Notice further that the cell $(H1, H3)$ is empty because it is neither incompatible nor has associated transition pairs.

Next, the table is resolved iteratively.

2. At each step of the iteration every state pair that has not yet been determined as F is updated as follows: If the cell of a state pair (p, q) includes a transition pair (p', q') whose cell has already been determined as F (incompatible), then the cell of (p, q) is also denoted F . Otherwise, the cell of (p, q) is modified as follows: Each transition pair (p', q') in the cell of (p, q) is replaced by all the transition pairs that appear in the cell of (p', q') .
3. If in a given iteration step no new incompatible state pairs are found (i.e., no new F designations are added to the table), then all the state pairs that are not designated as F , are given the designation T (for true). This completes the table resolution procedure and the determination of all compatible pairs.

To illustrate the iteration steps of the procedure, let us return to our transmission example. The table of Figure 8 is obtained from that of Figure 7 as follows: First we replace the transition pairs in the cell $(L1, L2)$ by those in the cell $(L2, L3)$. The cells $(L1, L3)$ and $(L2, L3)$ are denoted with F because their cells include incompatible pairs. The remaining undecided state pairs (those that have not yet been given the value F) are modified according to the algorithmic procedure. For example, in the cell $(M1, M2)$ we list the transition pairs from the table of Figure 7 of the cell $(H1, H2)$ that consists of $(H2, H3)$.

L2	$(L2,L3)_\gamma$						
	$(M1,H1)_\beta$						
L3	F	F					
M1	F	F	F				
M2	F	F	F	$(H2,H3)_\delta$			
H1	F	F	F	F	F		
H2	F	F	F	F	F	$(H1,H2)_\gamma$	
H3	F	F	F	F	F		$(H2,H3)_\delta$
	L1	L2	L3	M1	M2	H1	H2

Figure 8. Resolution table (after first iteration).

In the next resolution step the table of Figure 9 is obtained. Here the cell $(L1,L2)$ is marked F upon substituting the value F of the cell $(M1,H1)$ which is incompatible. The remaining undecided cells are modified as specified by the algorithm. In fact, notice that no further change needs to be made to the table.

L2	F						
L3	F	F					
M1	F	F	F				
M2	F	F	F	$(H2,H3)_\delta$			
H1	F	F	F	F	F		
H2	F	F	F	F	F	$(H1,H2)_\gamma$	
H3	F	F	F	F	F		$(H2,H3)_\delta$
	L1	L2	L3	M1	M2	H1	H2

Figure 9. Resolution table (after second iteration).

In the next step, no further incompatible pairs are created and the table remains identical to that of Figure 9. At this point, all the remaining undecided cells are marked T as shown in the table of Figure 10, concluding the table resolution.

Thus, as seen in Figure 10, for the example of Figure 1, the set of compatible pairs consists of $(M1, M2)$, $(H1, H2)$, $(H1, H3)$, and $(H2, H3)$. Notice that the states $L1$, $L2$ and $L3$ do not appear in any compatible pairs and therefore the singleton sets $(L1)$, $(L2)$ and $(L3)$ are clearly maximal compatibles.

L2	F						
L3	F	F					
M1	F	F	F				
M2	F	F	F	T			
H1	F	F	F	F	F		
H2	F	F	F	F	F	T	
H3	F	F	F	F	F	T	T
	L1	L2	L3	M1	M2	H1	H2

Figure 10. Resolution table (completed).

Generation of the set of maximal compatibles

The procedure for generation of maximal compatibles consists of first systematically creating all compatible sets. We begin by computing all compatible triples, then compatible quadruples, then quintuples, and so on. A compatible triple is a triple all three of whose pairs are compatible; a compatible quadruple is a quadruple all of whose pairs are compatible, which is equivalent to a quadruple whose four triples are all compatible, and so on. Once all compatibles are listed, the

maximal ones can easily be computed by deleting from the list all compatibles that are contained within larger ones.

For the transmission example, the maximal compatibles are easily found to be the sets $(L1)$, $(L2)$, $(L3)$, $(M1,M2)$ and $(H1,H2,H3)$. It is also not difficult to see that, in this case, they partition the state set into disjoint subsets and hence form the (unique) minimal cover by maximal compatibles.

Generation of reduced models

The generation of a reduced model that can serve as a correct user model for the given machine and specification is based on an abstraction of the machine-model. This reduced model is obtained by clustering the states into sets that consist of a minimal cover by maximal compatibles.

To this end, let us assume that a minimum cover consists of a given set of maximal compatibles C_1, \dots, C_l , where the set C_i , $i = 1, \dots, l$, consists of states $\{q_{i_1}, \dots, q_{i_{n_i}}\}$ of the machine model. The maximal compatibles C_1, \dots, C_l form the state set of the reduced model. Here it is noteworthy that a minimal cover by maximal compatibles need not be a partition of the state set into disjoint subsets. Specifically, while each state of the machine model must be contained within some maximal compatible set, it may well be the case that a state is contained in more than one maximal compatible of the minimal cover. That is, these sets may (sometimes) have overlaps.

Next, we turn to computing the transitions in the reduced model. An event symbol σ is said to be active at C_i , if there exists an outgoing transition in the machine model labeled by σ , at some state $q \in C_i$. That is, there exists a state q' in the machine model, such that $q \xrightarrow{\sigma} q'$ is defined. We denote by $C_i(\sigma)$ the set of all states $q \in C_i$ for which an outgoing transition labeled by σ exists.

Next, we define $S_i(\sigma)$ to be the set of all states q' of the machine model, such that $q \xrightarrow{\sigma} q'$ for some $q \in C_i(\sigma)$. Thus, the set $S_i(\sigma)$ is the set of all states of the machine model that can be reached from states in C_i through the event σ . It readily follows from the definition of compatible sets that there exists one or more element of C_1, \dots, C_l which contain $S_i(\sigma)$. In the reduced model we then create a transition labeled by σ going from the state C_i to the state C_j , where C_j is the maximal compatible that contains $S_i(\sigma)$. If more than one such set C_j exists, we can choose any one of these (and to avoid non-determinism in the reduced model we choose exactly one).

To summarize, the reduced model associated with the minimal cover C_1, \dots, C_l is obtained as follows. The state set of the reduced model consists of elements p_1, \dots, p_l (think of p_i as associated with C_i). There is a transition labeled σ from p_i to p_j if C_j is the (chosen) set that contains $S_i(\sigma)$. The reduced model is initialized at state p_k if the machine model is initialized at a state in C_k (where, as before, there may be more than one possible selection if the initialization state is contained in more than one of the C_i).

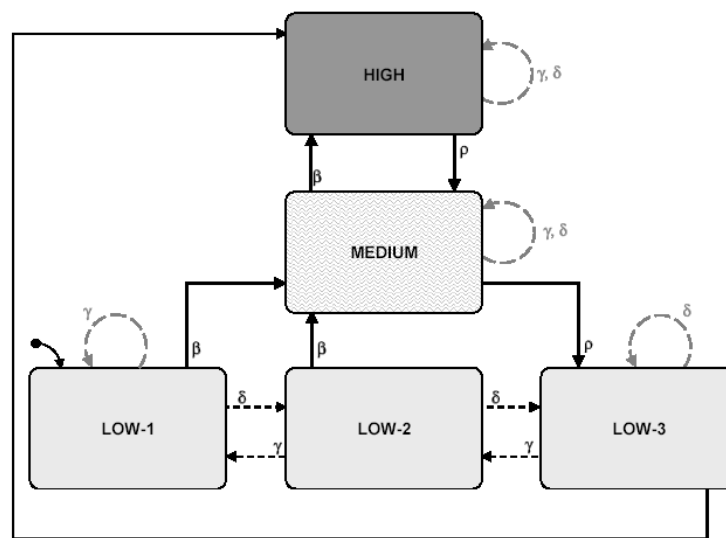


Figure 11. The reduced user model.

The reduced model obtained for the transmission example is shown in Figure 11. The correctness of this reduced model as a user model for the specification is verified in Figure 12 in which the composite model with the machine model of the transmission is displayed.

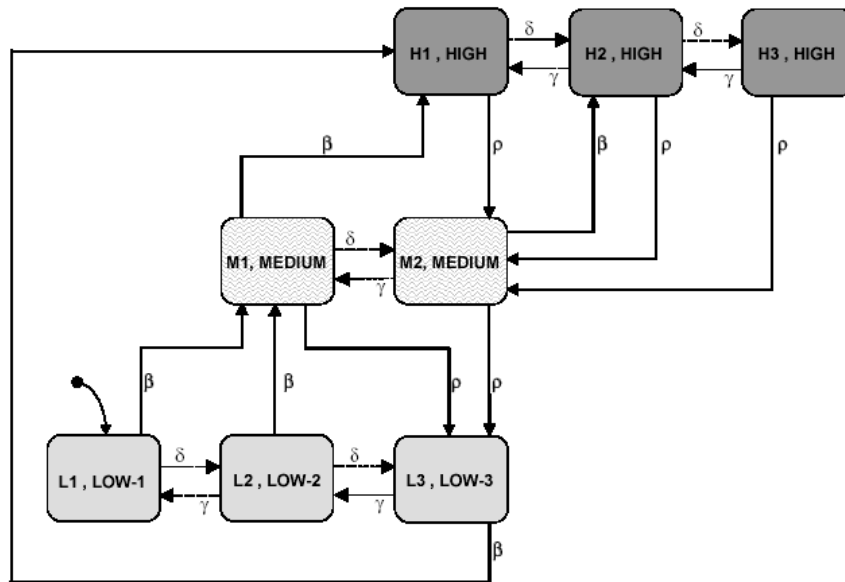


Figure 12. Verification of the reduced model (no error states and no blocking are detected).

Event Abstraction

The final step of the model reduction procedure consists of the abstraction of the reduced model's event set (when possible). Specifically, we ask which events can be internalized (i.e., need not be monitored) and which events can be clustered into groups so that instead of being monitored individually, they be monitored collectively. That is, the user will be informed that some events in the group occurred, but will not be informed which events of the group actually took place.

To this end the following abstraction rules apply:

1. An event can be internalized if it occurs in the reduced model only in self-loops.

2. A set of events can be grouped together, if every state transition that can be triggered by any event of the group can also be triggered by any other event of the group.

In the transmission example no event abstractions are possible. An illustration of event abstractions is provided in the example of the next section.

AN ABSTRACT MACHINE EXAMPLE

In the above discussion on verification and machine model reduction, we used an example of a transmission system. In this final section, we shall apply the reduction algorithm to a somewhat more complex machine. The machine in Figure 13 has 9 states and 25 transitions. There are three specification classes: the gray region that includes states 7, 8, and 9; the wave-like region that harbors state 4 and 6; and the rest of the states of the machine (1, 2, 3, and 5). The task specification is similar to our previous one: the user has to track the machine along these three regions (or modes). Specifically, the user must be able to identify the current mode of the machine and anticipate the next mode of the machine as a consequence of his or her interactions.

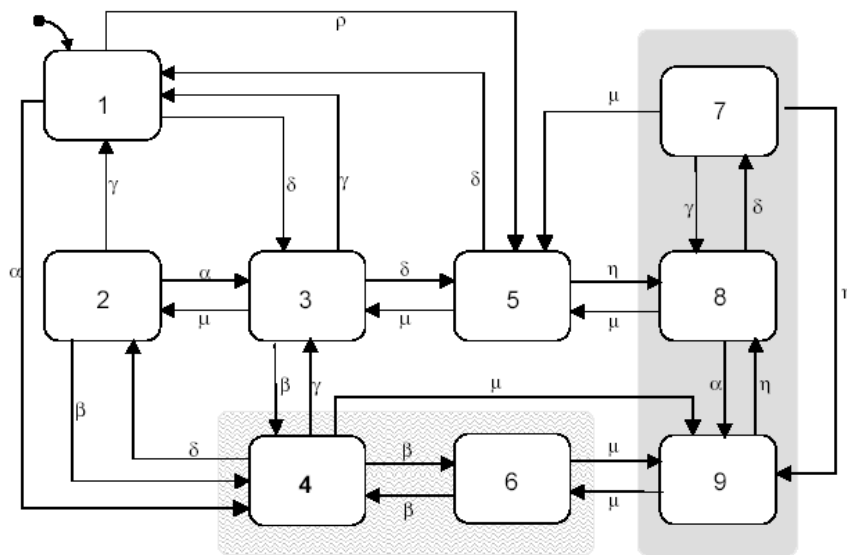


Figure 13. An abstract machine model.

We perform the reduction procedure along the steps described in the previous section. First the table is constructed, and then the iterations are performed. The procedure terminates with only one minimal cover of maximal compatibles that consists of four state sets: (1,3,5) (2,3,5) (4,6) (7,8) and (9). Notice however, that this example illustrates a case in which the cover is not a partition of the state set. Indeed, the state 3 is included in two distinct maximal compatibles.

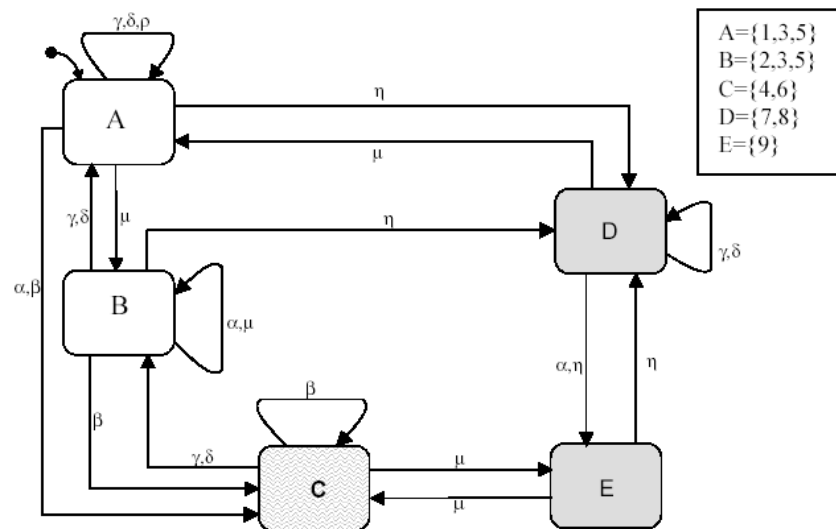


Figure 14. Reduced model.

We then arbitrarily assign names to these sets, and call them A, B, C, D, and E, respectively. The reduced machine is obtained upon computation of the abstracted transitions as explained earlier, and is shown in Figure 14. It can be seen in this figure that the event ρ occurs only in the self-loop in state A and that the events γ and δ are interchangeable. Thus, ρ can be internalized and the events γ and δ can be grouped. The result of this event abstraction is presented in the final reduced (user) model of Figure 15, which contains only 5 states and 16 transitions. The verification result of this model is presented in Figure 16. No error states or blocking are detected.

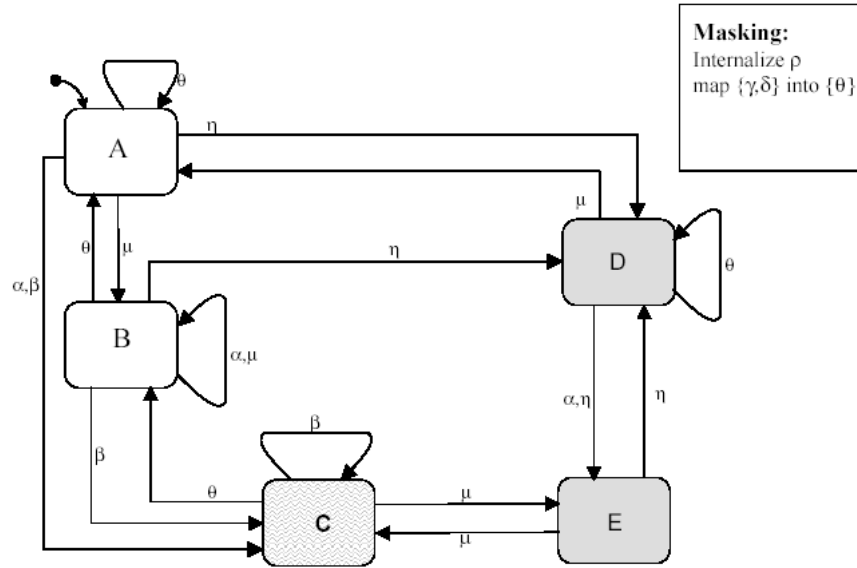


Figure 15. Reduced model (with masking and internalization of event).

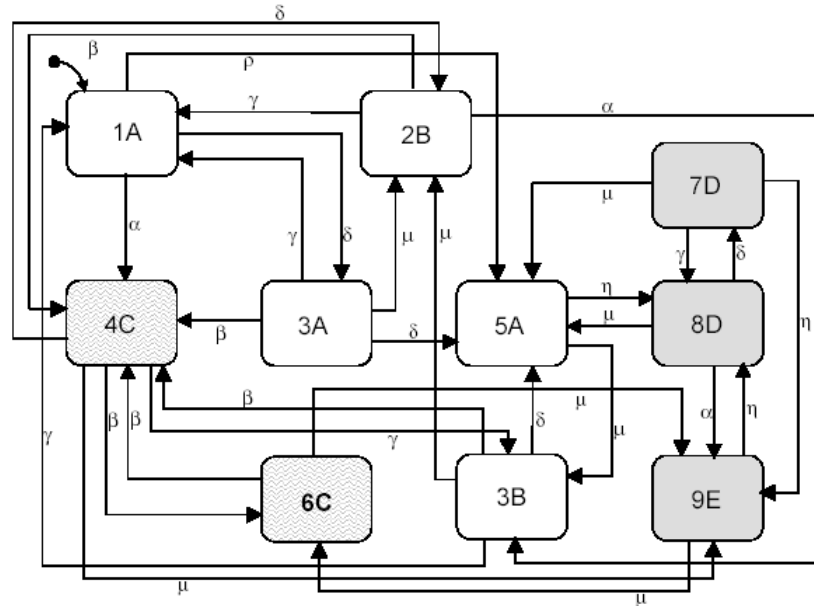


Figure 16. Composite model.

CONCLUSIONS

In this report we discussed several formal aspects of the design of human-automation interaction. Specifically, we focused attention on the construction and verification of correctness of user models and interfaces. Two objectives guided us in our design and analysis: (1) that the interfaces and user models be correct; and (2), that they be as simple as possible. We have described a systematic procedure for generating such correct and succinct user-models and interfaces.

The discussion and the examples illustrate that even for machines that are seemingly simple, i.e., that have very few states and straightforward task specifications, finding a correct interface and user-model is not a trivial matter. Interfaces that intuitively may appear to be correct are shown, after applying formal verification, to be faulty. It is therefore not surprising that we encounter so many automation problems in commonly encountered systems. Indeed, such problems can be found in almost every computer-based system.

Thus, the main focus of the report is on a systematic procedure for constructing correct and succinct user-models and interfaces. The proposed reduction procedure generates interfaces that are not necessarily intuitive or easily correlated with the underlying system (e.g., see the reduced user model of Figure 15). Nevertheless, these user models are formally correct and efficient. They are also, irreducible. This is a marked departure from the usual *ad hoc* and “straightforward” way of constructing abstractions in interface design. But this change in approach is necessary, given the complexity of current systems, the expected increase in complexity of future systems, and the ever-increasing requirements for correct and reliable operation.

As discussed in the section “compatible states sets and covers,” the proposed procedure may lead to more than one possible minimal (irreducible) interface and user-model. That is, it may find

several minimal covers (of maximal compatibles). These minimal covers are all correct and efficient reductions of the same machine and task-specification. Naturally, the decision as to which one is selected constitutes a human-factors and/or engineering design decision. It affords the designer with several candidate interfaces and allows designers the freedom to choose the most appropriate one, given other design considerations such as Graphical User Interface considerations, users' preferences, and ease of implementation.

While the discussion and examples have focused on discrete-event systems and finite state machine representations, the approach is amenable to other type of representations. It remains, however, an interesting topic of future research, to expand the approach to systems that have continuous and discrete events (hybrid systems) as well as timed systems.

ACKNOWLEDGMENTS

This work was conducted as part of NASA's base research and technology effort, human-automation theory sub-element (RTOP #548-40-12). 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 model reduction software. The authors also thank George Meyer and Rowena Morrison for their helpful insights.

REFERENCES

- Abbott, K., Slotte, S. M., and Stimson, D. K. (1996). *The interface between flightcrews and modern flight deck systems*. Washington, DC: Federal Aviation Administration.
- Andre, A. & Degani, A. (1997) Do you know what mode you're in? An analysis of mode error in everyday things. In Mouloua, M. & Koonce, J.M., (Eds.) *Human-automation interaction: Research & Practice*, Mahwah, NJ: Lawrence Erlbaum.
- Degani, A. and Heymann, M. (in press). Formal Verification Of Human-Automation Interaction. *Human Factors*.
- Heymann M., and Degani A. (2001). *On formal abstraction and verification of human-machine interfaces: the discrete event case*. NASA Technical Memorandum.
- Federal Aviation Regulation 25.1329. *Certification of Flight Guidance Systems*. Washington, DC: Code of Federal Regulations.
- Kohavi, Z. (1978). *Switching and Finite Automata Theory*. New York: McGraw-Hill.
- 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.
- Parasuraman, R., Sheridan, T.B., and 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(3), 286-297.
- Paull, M.C. and 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.
- Woods, D., Sarter, N., and Billings, C. (1997). Automation surprises. In G. Salvendy (Ed.), *Handbook of human factors and ergonomics* (pp. 1926-1943). New York: John Wiley.

LIST OF FIGURES AND TABLES – TO BE DONE

Figure 1. Transmission system. Figure 2. Proposed interface and user model.

Figure 3. Alternate interface and user model.

Figure 4. Masked synchronous composition.

Figure 5. Composite model of the alternative user model.

Figure 6. Table of all pairs.

Figure 7. Resolution table (initial).

Figure 8. Resolution table (after first iteration).

Figure 9. Resolution table (after second iteration).

Figure 10. Resolution table (completed).

Figure 11. The reduced user model.

Figure 12. Verification of the reduced model. Figure 13. An abstract machine model.

Figure 14. Reduced model.

Figure 15. Reduced model.

Figure 16. Composite model.

ASAF DEGANI and MICHAEL HEYMANN (*On the design of human-automation interfaces: A formal approach*)

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 Division at NASA Ames. His primary research topics are in the areas of human interaction with automation, modeling, design of procedures and decision aids, cockpit displays, and aviation safety.

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 30 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.