Designing Automation to Reduce Operator Errors

Nancy G. Leveson
Computer Science and Engineering
University of Washington

Everett Palmer
NASA Ames Research Center

Introduction

Advanced automation has been accompanied, particularly in aircraft, with a proliferation of modes, where modes define mutually exclusive sets of system behavior. The new mode-rich systems provide flexibility and enhanced capabilities, but they also increase the need for and difficulty of maintaining mode awareness. While automation has eliminated some types of operator mode-awareness errors, it has also created the potential for new types of mode-related problems [SW95].

After studying accidents and incidents in the new, highly automated aircraft, Sarter and Woods have concluded that certain errors are non-random and predictable [SW95]: They are the regular and predictable consequences of a variety of identifiable factors. Although these errors are accentuated by poor interface design and gaps or misconceptions in the user's mental model of the system, an important factor is inconsistent automation behavior.

Sarter and Woods have identified some of these predictable error forms. Leveson et. al. [LPS97] and Degani [Deg96] have defined taxonomies of automation features that lead to mode confusion. This paper describes an approach to dealing with mode confusion errors by first modeling blackbox software behavior and then using analysis methods and tools to assist in searching the models for predictable error forms, i.e., for features that contribute to operator mistakes. The analysis results can be used to redesign the automation, to change operator training and procedures, or to design appropriate human-computer interfaces to help avoid mistakes.

The approach requires a model of the blackbox behavior that is both formal and easily readable and reviewable by humans. The models we use are part of the software specifications in a methodology called SpecTRM (Specification Tools and Requirements Methodology) and thus the analysis is done directly on the system requirements specification and does not require extra modeling effort. SpecTRM includes a suite of analysis tools to detect errors and potentially hazardous behavior early in system development when tradeoffs and changes can more easily be made. In addition to providing design guidance, this approach might provide a way of "measuring" or evaluating the cognitive demands involved in working with specific automated devices. Hansman has suggested that automation complexity be defined in terms of the predictability of the automation behavior [Hans97]. This predictability can potentially be evaluated on the formal SpecTRM-RL (SpecTRM Requirements Language) models.

The rest of the paper provides more information about the approach and illustrates its use on a commonly reported mode confusion error called a "kill-the-capture" bust.

Mode Confusion Analysis

Most accidents related to software behavior can be traced back to errors or omissions in the software requirements, not to implementation or coding errors [Lev95, Lut93]. Although a great deal of effort has been expended in software engineering on finding software design and implementation errors, much less has been accomplished in terms of validating requirements specifications beyond executing them for a few test cases or showing the consistency of a formal specification with various properties of the underlying mathematical model [HL96, HLK95]. Most of the specfication errors and omissions that lead to accidents are unlikely to be found using these techniques. The testing of any complex software is necessarily very incomplete, and consistency with a mathematical model does not imply consistency with required properties of a real world application.

To deal with this problem, we have specified a set of criteria for completeness and correctness of blackbox process-control requirements specifications that are related to safety [JLHM91, Lev95]. These criteria were derived using real accidents and industrial experience with process-control software, and they have been validated by experimental application to the NASA Gallileo and Voyager software [Lut92, Lut93] and through industrial use. We are contining to extend the criteria, most recently with the goal of reducing mode confusion errors, and to validate them on real software [MLR97].

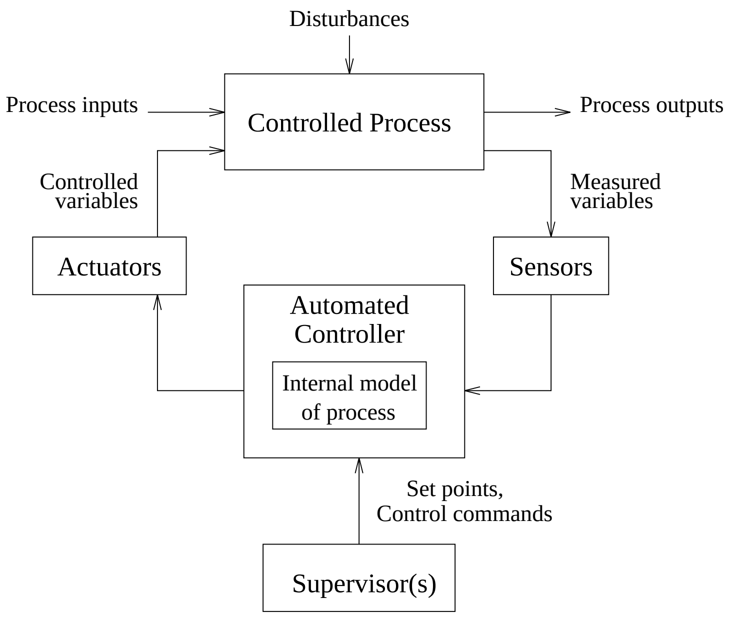


Figure 1: Simple Control Loop Model

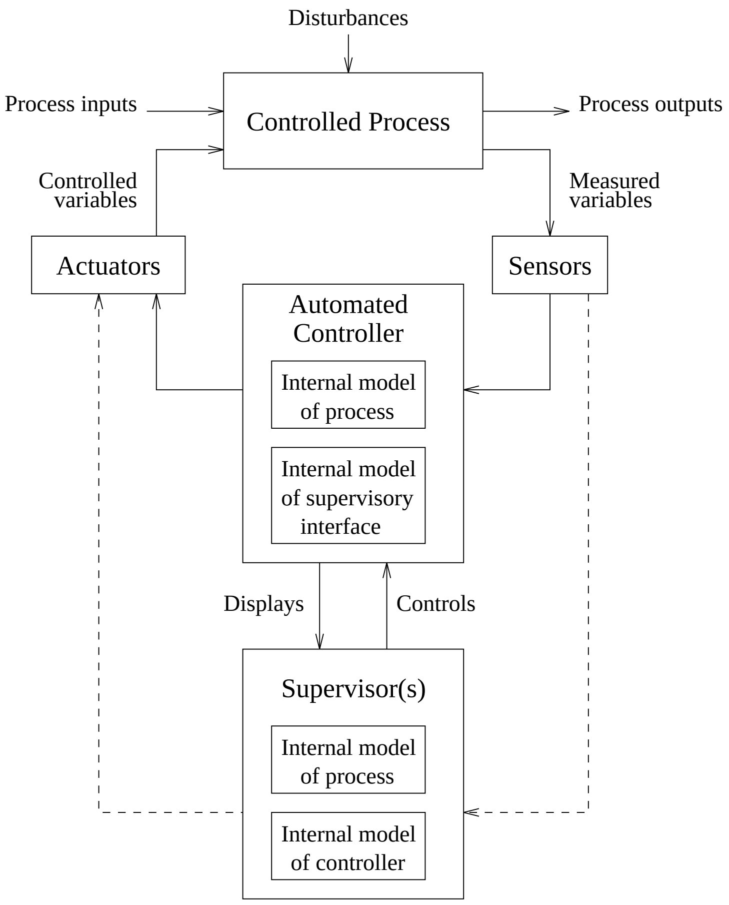


Figure 2: Modified Model to Account for Operator Error and Mode Confusion.

To apply the criteria, a blackbox state-machine model of the automation behavior is required. Blackbox requirements specifications do not contain information about internal design (the software design if the automated controller is a computer) but are written strictly in terms of externally visible inputs and outputs and the effects of these on a model of the process being controlled (see Figure 1). The process model is based on:

1. Current process state inferred from measured variables,
2. Past measured and inferred process states and variables,
3. Past outputs to actuators, and
4. Prediction of future states of the controlled process.

Accidents related to requirements (behavioral) specification occur when the internal model of the process becomes inconsistent with the state of the controlled process. This inconsistency may result from an incorrect model being specified originally (e.g., the model does not include basic required behavior for unusual or infrequently occurring cases) or from the modeled system state being updated incorrectly during execution, perhaps as a result of incorrect input from the sensors.

To define criteria related to mode confusion, we need to add a model of the controller-software interface to the automated controller. We also need to consider the supervisors' internal models of the expected behavior and state of the process and of the automated controller (see Figure 2). Accidents in this extended model may result from any of these models being incorrect or becoming inconsistent with the true state of the controlled process, the automated controller, or the supervisory interface (the human–computer interface). That is, accidents may result if any of the models are or become inconsistent with the state of the thing they are modeling and decisions or actions are made on the basis of the incorrect model. Criteria for correctness and safety can be specified in terms of these formal models and checked for particular system specifications.

Of course, we are not suggesting that it is possible to specify human mental models. Each person may have a different mental model of the system and the automation, and these may change over time within the same person. In fact, operators have been found to be able to function with multiple and inconsistent models [Luc87]. However, it is possible to state some high-level abstractions about required features of correct operator mental models—for example, that particular actions on the part of the operator will result eventually in particular changes in the automation and/or the system.

Note that we assume here that the operator's models are correct. This assumption will obviously not always be true. However, our approach involves first eliminating hazards for the ideal case. Then various types of hazard analysis can be used to determine which types of erroneous models will have the most serious consequences. The resulting information can be used for automation design, interface design, and operator training.

A previous paper described six categories of potential design flaws that can lead to mode confusion errors: interface interpretation flaws, inconsistent behavior, indirect mode changes, operator authority limits, unintended side effects, and lack of appropriate feedback [LPS97]. The rest of this paper shows an example of this approach for one particular common cause of mode confusion error, i.e., indirect mode changes. The basic criteria and analysis technique is being specified formally [Lev97], but we include only an informal description here.

Indirect Mode Change Example

Indirect mode changes occur when the automation changes mode without an explicit instruction by the operator. Such transitions may be triggered on conditions in the controller (such as preprogrammed envelope protection) or sensor input about the state of the controlled system (such as achievement of a target value). Indirect mode transitions create the potential for mode confusion and inadvertent activation of modes by the human controller. For example, the human controller may not update his or her models of the state of the process and the state of the automation and, based on these now incorrect models, issue an incorrect control command or fail to issue a required command.

An example of an accident that has been attributed to an indirect mode change occurred while an A-320 was landing in Bangalore. In this case, the pilot selection of a lower altitude while the automation was in the ALTITUDE ACQUISITION mode resulted in the activation of the OPEN DESCENT mode. It has been speculated that the pilots did not notice the mode annunciation because the indirect mode change occurred during approach when the pilots were busy and they were not expecting the change [SW95]. Another example of such an indirect mode change in the A-320 automation involves an automatic mode transition triggered when the airspeed exceeds a predefined limit. For example, if the pilot selects a very high vertical speed that results in the airspeed decreasing below a particular limit, the automation will change to the OPEN CLIMB mode, which allows the airplane to regain speed.

Palmer has described another example of a common indirect mode transition problem called a “kill-the-capture bust” that has been noted in many ASRS reports [Pal96]. Here we show the relevant parts of a SpecTRM-RL specification of the MD-88 control logic and describe how the problem can be detected and fixed.

	Thrust	Arm	Roll	Pitch
a. Level at 2100 ft.	SPD 186		VOR CAP	ALT HLD
b. Enter 5000 ft.	SPD 186	ALT	VOR CAP	ALT HLD
c. Set VERT/SPD	SPD 186	ALT	VOR CAP	VERT SPD
d. Enter 255	SPD 255	ALT	VOR CAP	VERT SPD
e. Approaching 4000 ft.	SPD 255	ALT	VOR TRK	VERT SPD
f. Push IAS	CLMP	ALT	VOR TRK	IAS
g. Automatic altitude capture	SPD 255		VOR TRK	ALT CAP
h. Adjust vertical speed	SPD 255		VOR TRK	VERT SPD

Figure 3: Flight Mode Annunciator (FMA) Displays for the Example Incident

In the incident, the crew had just made a missed approach and had climbed to and leveled at 2,100 feet. Figure 3 shows the sequence of Flight Mode Annunciator (FMA) values during the incident. The crew received the clearance to “...climb now and maintain 5,000 feet ...”. The Captain set the desired altitude to 5,000 feet, set the autopilot pitch mode to vertical speed with a value of approximately 2,000 feet per minute and the autothrottle to SPD mode with a value of 256 knots (Figure 3(c) and (d)). Climbing through 3,500 feet, the Captain called for flaps up, and at 4,000 feet he called for slats retract and pushed the IAS button (Figure 3(f)). The pitch mode became IAS, and the autothrottle went to CLAMP mode. At this point, altitude capture was still armed. Three seconds later, the autopilot automatically switched to altitude capture mode. The arm window went blank, and the pitch window showed ALT CAP (Figure 3(g)). A tenth of a second later, the Captain adjusted the vertical speed wheel to a value of about 4,000 feet per minute. This speed adjustment caused the pitch autopilot mode to switch from altitude capture to vertical speed (Figure 3(h)). Climbing through 4,500 feet, the FMA was as shown in Figure 3(h), and the approaching altitude light was on. As the altitude passed through 5,000 feet at a vertical velocity of about 4,000 feet per minute, the Captain remarked, “Five thousand. Oops, it didn’t arm.” He pushed the ALT HOLD button and switched off the autothrottle. The aircraft continued to climb to about

5,500 feet and the ALTITUDE–ALTITUDE voice warning sounded repeatedly.

To identify and fix the problem, we use a formal model. A SpecTRM-RL model has two parts: a graphical model of the state machine and a specification of the logic on the transitions. Figure 4 shows part of the graphical SpecTRM-RL state machine model of the MD-88 vertical control logic needed to understand the incident and how to fix the software to avoid it. In order to keep the model small enough to fit in the paper, only parts of it are shown but during system engineering a complete model would be constructed. The graphical model has three main parts: the input–output interface (where the supervisory interface is one part), the operating modes of the automation itself (in this case the autoflight system), and the process model which includes both the process (aircraft) operating modes and models of the aircraft components.

In the supervisory interface, square boxes denote inputs and outputs having finite state values. Circles represent numbers. Note that this model represents the automated controller's view of the state of the interface, not necessarily the real state of the controls and displays. A complete safety analysis would evaluate if and how discrepancies between the two could occur and also whether such discrepancies could lead to hazardous system states.

The state transition logic is specified in SpecTRM-RL using a form of logic tables we call AND/OR tables. A transition can be taken if any of the columns of the table evaluates to *true*. A column evaluates to *true* if all the (non-blank) rows in a column are true. Figure 5 shows the relevant transition logic for the example.

The problem occurs because the transition to ALT CAP mode results in a transition of the capture mode to UNARMED before the altitude has actually been acquired. Although this is annunciated to the pilot by the Arm annunciator changing to blank when pitch mode changes to ALT CAP, the absence of an indicator is well known to be an error-prone way to notify the pilot of a mode change.

How could this be detected from examining the logic? In general, an indirect mode change is one that occurs without an explicit pilot action to change the mode. The vertical control logic for the example has three mode transitions that do not require direct pilot input: (1) the transition from ANY to ALT CAP, (2) the second column of the transition from ANY to ALT HOLD, i.e, when the altitude is acquired and the pitch is in mode ALT CAP, and (3) the second column of the transition from ARMED to NOT ARMED. Each of these mode transitions is triggered by a change in a controlled system variable or by internal mode change within the automation.

We will assume that the pilot's mental model includes a cause and effect relationship between arming the altitude capture and eventually (although it may not be immediately) acquiring that altitude and holding it:

$$\text{set altitude and pull ALT} \rightarrow \ldots \rightarrow \text{ALT HOLD}.$$

Formal analysis will show that there is a path through the logic starting with the pilot pulling the altitude knob that does not result in the ALT HOLD state (specifically, this occurs when the automation is in the modes NOT ARMED and ALT CAP and the pilot does something that changes the pitch mode, in this case adjusting the vertical speed wheel).

One way to fix the problem is to change the transition logic to that shown in Figure 6. Note that although the second column of the transition table from ANY to ALT HOLD still does not require direct pilot input, the transition is not indirect by our definition because it satisfies the pilot model of the transition logic above. The transition from ANY to ALT CAP is still indirect, but there is no longer a path through the vertical control logic that violates the expected cause and effect relation between arming the capture and capturing the altitude when it is acquired. We note that this solution may violate other goals or desired behaviors of the autoflight system—the designers would have to determine this when deciding what solution to use. In addition, a more sophisticated solution may be required, e.g., a hysteresis factor may need to be added to the mode transition logic to avoid too rapid or "ping-ponging" transitions between pitch modes.

Finding indirect mode transitions does not mean the software must be changed. The identified criteria are simply clues for determining where to look for potential problems. The designers may decide that no real problem exists and make no changes or they may decide not to change the automation but instead to make changes in the interface design or in pilot training.

In general, it is not feasible to make all mode transitions direct in any sophisticated automated controller. The goal instead is to simplify the required pilot model of the automation behavior as much as possible. In this case, the pilot expects a direct mode transition from setting a target altitude and arming the altitude capture to eventually attaining capturing altitude, changing to ALT HOLD mode, canceling the ARM command. Any paths in the automation logic that will violate this assumption will be a source of potential mode confusion, even if the mode change is annunciated. Of course, the pilot may have more sophisticated knowledge of the automation logic and know that adjusting the vertical speed wheel will cancel the previously given altitude capture command. However, ,this knowledge assumes a much more

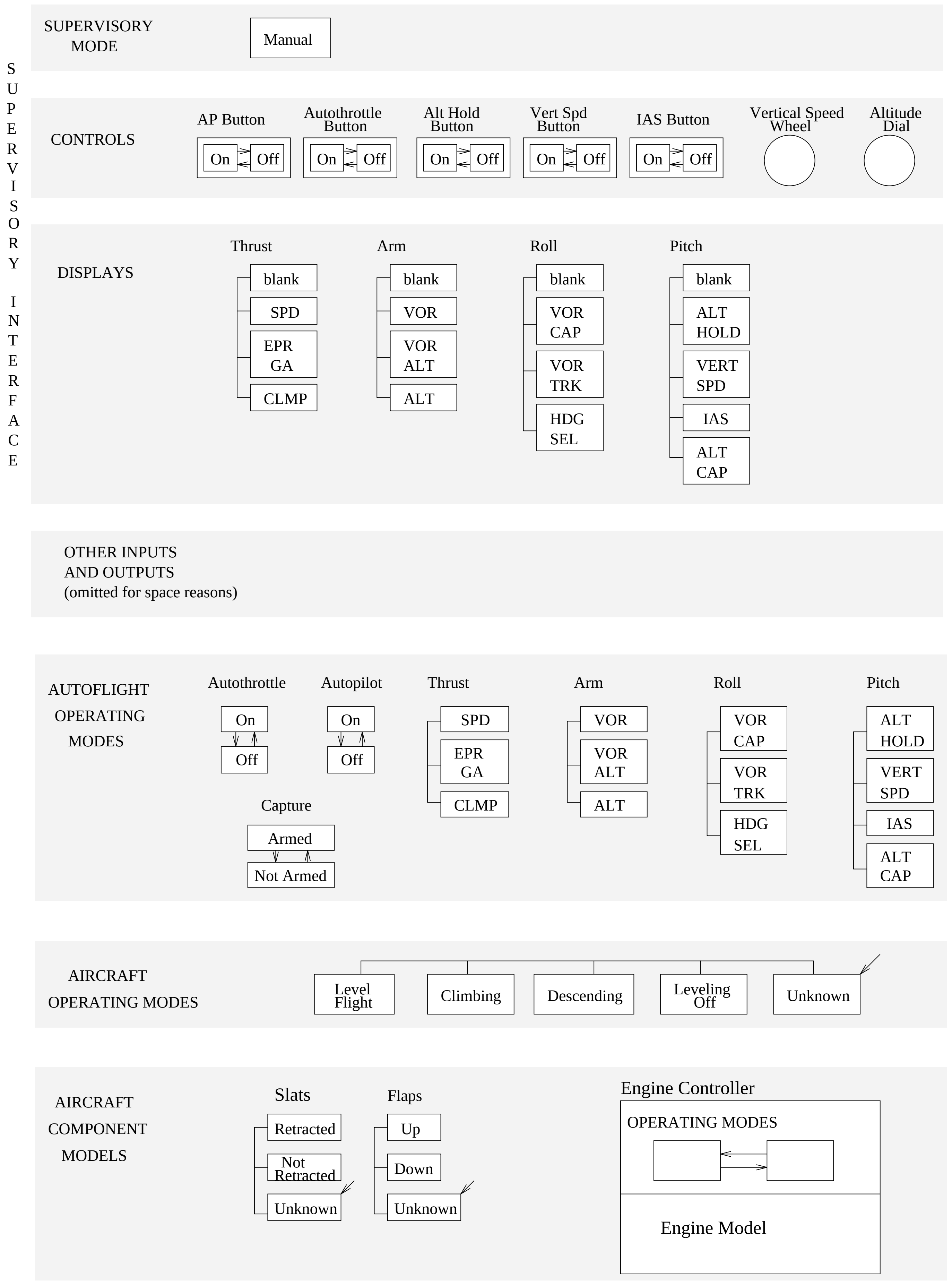
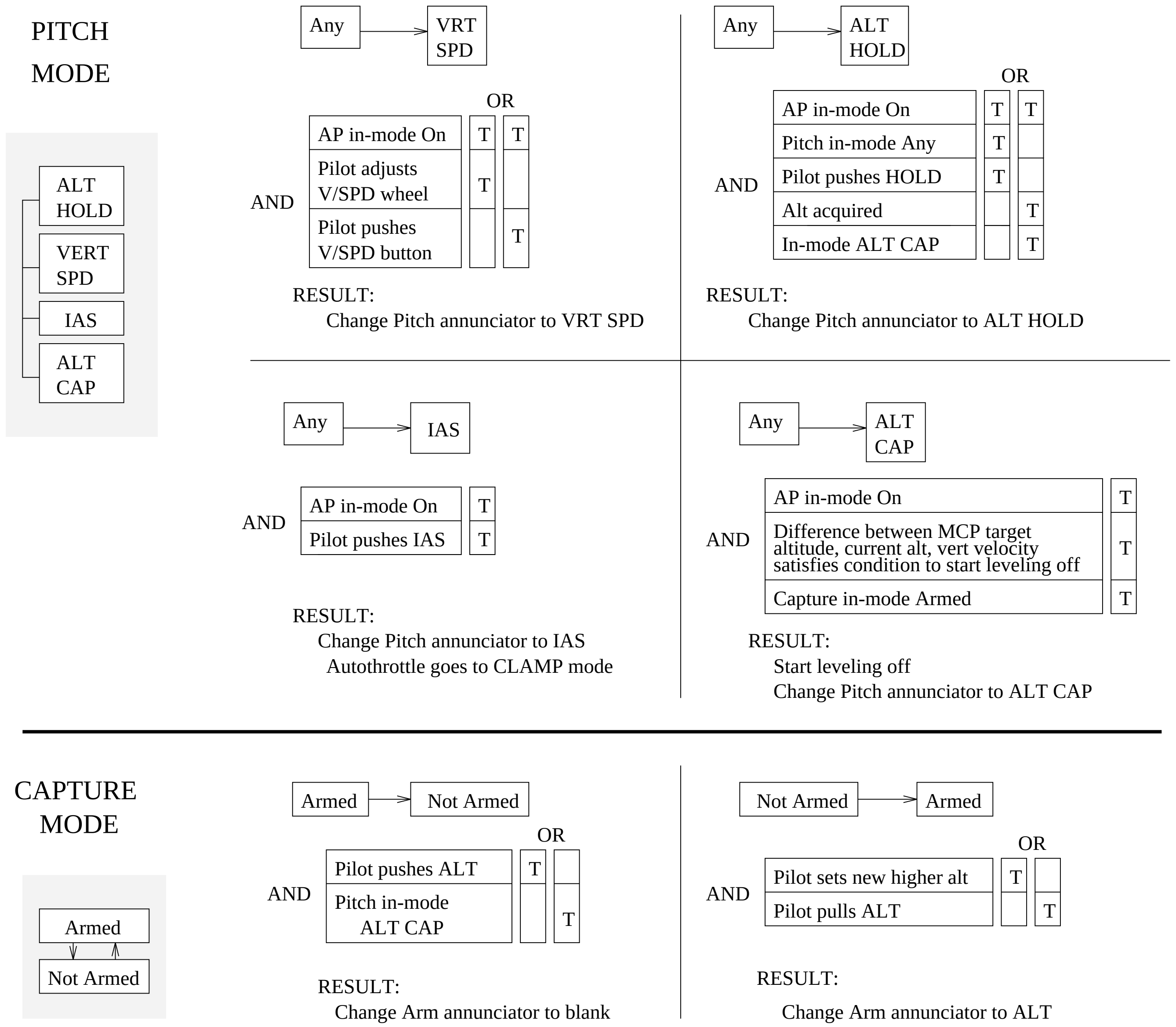


Figure 4: Part of the Graphical State Machine Model of the MD-88 Vertical Control Logic



AND	OR	
AP in-mode On	T	T
Pilot adjusts V/SPD wheel	T	
Pilot pushes V/SPD button		T

RESULT:
Change Pitch annunciator to VRT SPD

AND	OR	
AP in-mode On	T	T
Pitch in-mode Any	T	
Pilot pushes HOLD	T	
Alt acquired		T
In-mode ALT CAP		T

RESULT:
Change Pitch annunciator to ALT HOLD

AND	
AP in-mode On	T
Pilot pushes IAS	T

RESULT:
Change Pitch annunciator to IAS
Autothrottle goes to CLAMP mode

AND	
AP in-mode On	T
Difference between MCP target altitude, current alt, vert velocity satisfies condition to start leveling off	T
Capture in-mode Armed	T

RESULT:
Start leveling off
Change Pitch annunciator to ALT CAP

AND	OR	
Pilot pushes ALT	T	
Pitch in-mode ALT CAP		T

RESULT:
Change Arm annunciator to blank

AND	OR	
Pilot sets new higher alt	T	
Pilot pulls ALT		T

RESULT:
Change Arm annunciator to ALT

Figure 5: Transition Logic for the Pitch and Capture Modes

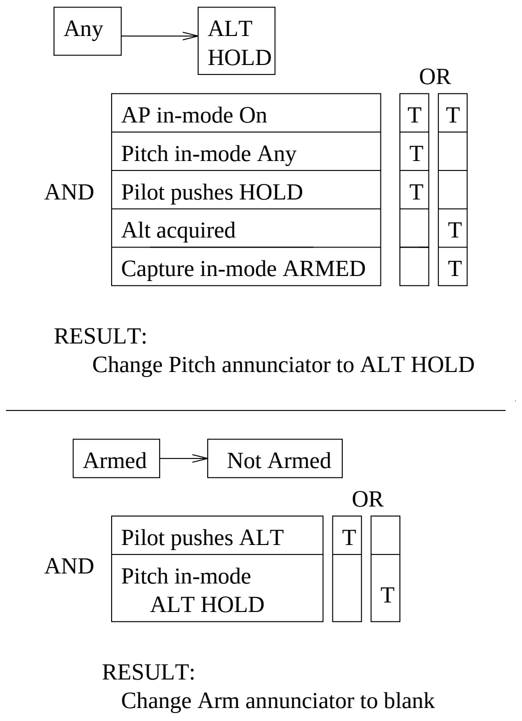


Figure 6: Revised Transition Logic for the Pitch and Capture Modes

complex model of automation behavior on the part of the pilot and makes the automation behavior more difficult to predict. The number of ASRS reports on this error leads us to believe that this assumption is not realistic.

Several caveats are important here. First, we are only guessing at the MC-88 software logic on the basis of the observed behavior. The real software specifications or code would have to be examined to determine what logic is actually implemented. Second, making the change we have recommended may not be feasible (or correct) due to other unmodeled parts of the logic that depend on the ARMED mode: A real development project would have the entire logic modeled and would be able to make the appropriate tradeoffs and design decisions.

REFERENCES

[Deg96] Degani, A. *Modeling Human-Machine Systems: On Modes, Error, and Patterns of Interaction.* Ph. D. thesis, Georgia Institute of Technology, 1996.

[Hans97] Hansman, John. Personal communication.

[HL96] Heimdahl, M. P. E. and N. Leveson. Completeness and consistency analysis of state-based requirements. *Transactions on Software Engineering*, June 1996.

[HLK95] Heitmeyer, C., Labaw, B., and Kiskis, D. Consistency checking of SCR-style requirements specifications. *Int. Symposium on Requirements Engineering*, York, 1995.

[JLHM91] Jaffe, M.S, Leveson, N.G., Heimdahl, M.P.E., and Melhart, B.E.. Software requirements analysis for real-time process-control systems. *IEEE Transactions on Software Engineering*, SE-17(3):241–258, March 1991.

[Lev95] Leveson, N.G. Safeware: System Safety and Computers. Addison-Wesley Publishing Co., 1995.

[Lev97] Leveson, Nancy G. Mode Confusion Modeling and Analysis, in preparation.

[LPS97] Leveson, N.G., Pinnell, L.D., Sandys, S.D., Koga, S., and Reese, J.D. Analyzing Software Specifications for Mode Confusion Potential. *Proc. Workshop on Human Error and System Development*, Glascow, March 1997.

[Luc87] Lucas, D.A. Mental models and new technology. *New Technology and Human Error*, John Wiley & Sons, 1987, pp. 337–340.

[Lut92] Lutz, R.R. Analyzing software requirements errors in safety-critical, embedded systems. *Software Requirements Conference*, 1992.

[Lut93] Lutz, R.R. Targeting safety-related errors during software requirements analysis. *Proc. Sigsoft '93: Foundations of Software Engineering*, 1993.

[MLR97] Modugno, F., Leveson, N.G., Reese, J.D., Partridge, K., and Sandys, S.D. Integrated Safety Analysis of Requirements Specifications. *Requirements Engineering Journal*, to appear.

[Pal96] Palmer, E. "oops, it didn't arm" – a case study of two automation surprises. NASA Technical Report, 1996.

[SW95] Sarter, N. D. and D. Woods "How in the world did I ever get into that mode?": Mode error and awareness in supervisory control. *Human Factors 37*, 5–19.

[SW95] Sarter, N. D. and D. Woods Strong, silent, and out-of-the-loop. CSEL Report 95-TR-01, Ohio State University, February 1995.