Presented at the 18th AIAA/IEEE Digital Avionics Systems Conference (DASC), St Louis, MO, October 1999.

An Automated Method To Detect Potential Mode Confusions[*]

John Rushby, SRI International, Menlo Park, California

Judith Crow, SRI International, Menlo Park, California

Everett Palmer, NASA Ames Research Center, Moffett Field, California

Introduction

Mode confusions are a type of "automation surprise"—circumstances where an automated system behaves differently than its operator expects. It is generally accepted that operators develop "mental models" for the behavior of automated systems and use these to guide their interaction with the systems concerned, so that an automation surprise results when the actual system behavior diverges from its operator's mental model.

Complex systems are often structured into "modes" (for example, an autopilot might have different modes for altitude capture, altitude hold, and so on), and their behavior can change significantly across different modes. "Mode confusion" arises when the system is in a different mode than that assumed by its operator; this is a rich source of automation surprises, since the operator may interact with the system according to a mental model that is inappropriate for its actual mode. Mode confusions have been implicated in several recent crashes and other incidents, and are a growing source of concern in modern automated cockpits.

If we accept that mode confusions are due to a mismatch between the actual behavior of a system and the mental model of its operator, then one way to look for potential mode confusions is to compare the design of the actual system against a mental model. There are two challenges here: how to get hold of a mental model, and how to do the comparison. Through observation, questionnaires, and other techniques, psychologists have been able to elicit the mental models of individual operators (typically pilots). However, comparison between a design and the mental model of a specific individual will provide only very specific information; we are interested in whether a design is *prone* to mode confusions, and for this purpose it is more useful to compare the design against a generic mental model rather than that of an individual. Such a generic model can be extracted from training material (one of the purposes, often implicit, of a training manual is to induce adequate mental models) or specified as an explicit requirement (e.g., "this button should behave like a toggle"). Cognitive studies provide two important insights on the nature of these models: first, that they can be represented compactly by mathematical structures called "state machines"; second, that they tend to be fairly simple (which can be explained by application of two canonical simplifications [3]).

The fact that mental models can be represented as state machines suggests a solution to the second challenge mentioned above—for designs can also be represented as state machines (this idea underlies modern design techniques, such as Statecharts), and there are automated methods for comparing the behaviors of one (finite) state machine against those of another. These methods are members of a class of formal techniques, known as "model checking," that are quite mature and are used routinely in hardware design and protocol analysis to explore properties of systems having many millions of states.

[*] This work was supported by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under contract F49620-95-C0044 and by the National Science Foundation under contract CCR-9509931.

An Example

We outline the proposed method using a "kill the capture" example reported by Palmer [6, Case 2].

The example is one of five altitude deviation scenarios observed during a NASA study in which twenty-two airline crews flew realistic two hour missions in DC-9 and MD-88 aircraft simulators. To follow the scenario, it is sufficient to understand that the autopilot can be instructed to cause the aircraft to climb or to hold a certain altitude through the setting of its "pitch mode." In `VSP` (Vertical Speed) mode the aircraft climbs at the rate set by the corresponding dial (e.g., 2,000 feet per minute); in `IAS` (Indicated Air Speed) mode, it climbs at whatever rate is consistent with holding the air speed set by another dial (e.g., 256 knots); in `HLD` (Altitude Hold) mode, it holds the current altitude. In addition, certain "capture modes" may be *armed*. If `ALT` (Altitude) capture is armed, the aircraft will only climb as far as the altitude set by the corresponding dial, at which point the pitch mode will change to `HLD`; if the capture mode is not armed, however, and the pitch mode is `VSP` or `IAS`, then the aircraft will continue climbing indefinitely. The behavior of this system is complicated by the existence of an `ALT CAP` (Altitude Capture) pitch mode, which is intended to provide smooth leveling off at the desired altitude. The `ALT CAP` pitch mode is entered automatically when the aircraft gets close to the desired altitude and the `ALT` capture mode is armed (do not confuse the `ALT CAP` *pitch* mode with the `ALT` *capture* mode). The `ALT CAP` pitch mode disarms the `ALT` capture mode and causes the plane to level off at the desired altitude, at which point it enters `HLD` pitch mode.

The following scenario description is slightly reworded from the original to fit the terminology used here.

> The crew had just made a missed approach and had climbed to and leveled at 2,100 feet. They received the clearance to "…climb now and maintain 5,000 feet…" The Captain set the MCP (Master Control Panel) altitude window to 5,000 feet (causing `ALT` capture mode to become armed), set the autopilot pitch mode to `VSP` with a value of approximately 2,000 ft. per minute and the autothrottle to SPD mode with a value of 256 knots. Climbing through 3,500 feet the Captain called for flaps up and at 4,000 feet he called for slats retract. Passing through 4000 feet, the Captain pushed the `IAS` button on the MCP. The pitch mode became `IAS` and the autothrottles went to CLAMP mode. The `ALT` capture mode was still armed. Three seconds later the autopilot automatically switched pitch mode to `ALT CAP`. The FMA (Flight Mode Annunciator) ARM window went from `ALT` to blank and the PITCH window showed `ALT CAP`. A tenth of a second later, the Captain adjusted the vertical speed wheel to a value of about 4,000 feet a minute. This caused the pitch autopilot to switch modes from `ALT CAP` to `VSP`. 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 MCP `HLD` button and switched off the autothrottle. The aircraft then leveled off at about 5,500 feet as the "*altitude—altitude*" voice warning sounded repeatedly.

To see how model checking techniques could reveal the existence of the surprise in this scenario, we first need to construct a mental model that a pilot might plausibly employ. A plausible generic model might embody the idea that the pitch mode controls *how* the aircraft climbs, and the capture mode controls whether there is a *limit* to the climb. Another plausible component of a generic model is that once capture mode is armed, it becomes disarmed only when the aircraft reaches the desired altitude (unless the pilot manually disarms it).

This mental model is described by the state machine in Figure 1, where `HLD`, `ALT`, `IAS`, and `VSP` represent the events where the corresponding buttons are pressed, and `arrive` represents the event of the aircraft reaching the target altitude. There are three states in this model, representing the (mutually exclusive) situations where the aircraft is in altitude hold, or has an altitude capture active, or neither of these. Pressing the `IAS` or `VSP` buttons has no effect (on

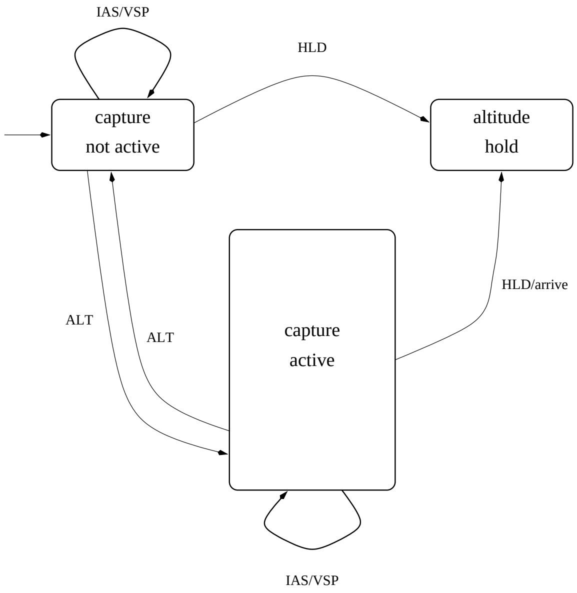

Figure 1: State Machine for Mental Model

the aspects of behavior considered here) when in either of the latter two states; pressing ALT causes each of these states to transition to the other; and pressing the HLD button causes a transition into altitude hold; arriving at the desired altitude also cases a transition into altitude hold when a capture is active. (To keep things simple, we do not model other behaviors, such as pressing the HLD button when already in altitude hold.)

The actual system is rather more complicated than the mental model: it has an ALT CAP pitch mode that is entered autonomously when a capture is active and the aircraft gets near the desired altitude. This behavior (also somewhat simplified) is described by the state machine in Figure 2

Since the mental model makes no mention of the ALT CAP pitch mode, it obviously differs from the actual system. This does not necessarily mean that the system harbors a surprise, however, because a mental model *should* suppress details considered unnecessary to understanding how to operate the system. The pilot might well be aware of the ALT CAP pitch mode and of its role in leveling the plane off—and may even be aware that the ALT CAP pitch mode and the ALT capture mode interact in some way—but could believe this is merely the implementation of the ideal capture mode assumed in the mental model. To

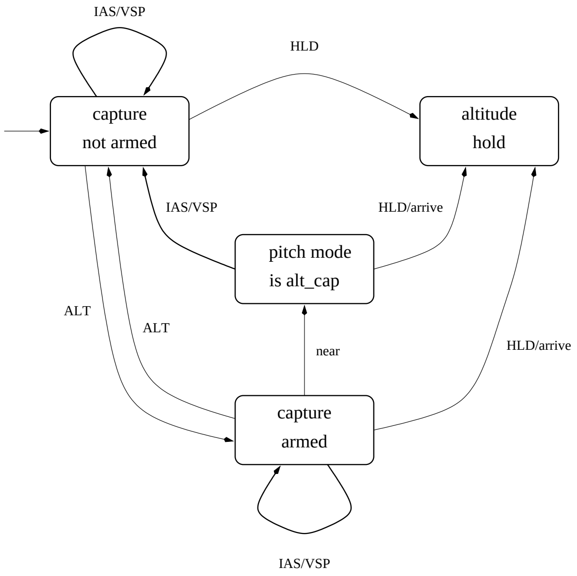

Figure 2: State Machine for Actual System

discover whether a surprise really does reside here, we need to “run” the state machines representing the actual system and the mental model on all possible sequences of inputs and compare their behavior.

In order to compare the behavior of the two state machines, we need to relate their states to each other. The actual system is flying a capture (in different ways) when in either of the two states in the lower center of Figure 2, so we can “abstract” these into a compound state that corresponds to the single “capture active” state of the mental model. This is shown in Figure 3.

Erasing the detail in the compound state, we arrive at the fully abstracted state machine for the actual system that is shown in Figure 4.

The states and events of this abstracted state machine correspond in the obvious way to those of the mental model, and any potential automation surprise will be manifested as a difference in their transition structures. Comparing Figures 1 and 4 we immediately see such a difference: the transition on VSP and IAS from the compound state of the abstracted actual system to the “capture not active” state. (Notice there are two transitions labelled IAS/VSP from the compound state; these represent nondeterministic choice at this level—we have to look inside the compound state to see which transition will actually occur.) This suggests that the sequence of events: VSP,

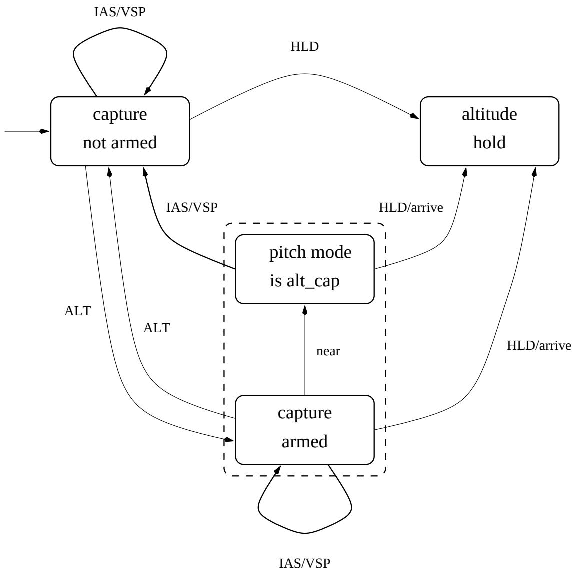

Figure 3: Partially Abstracted State Machine for Actual System

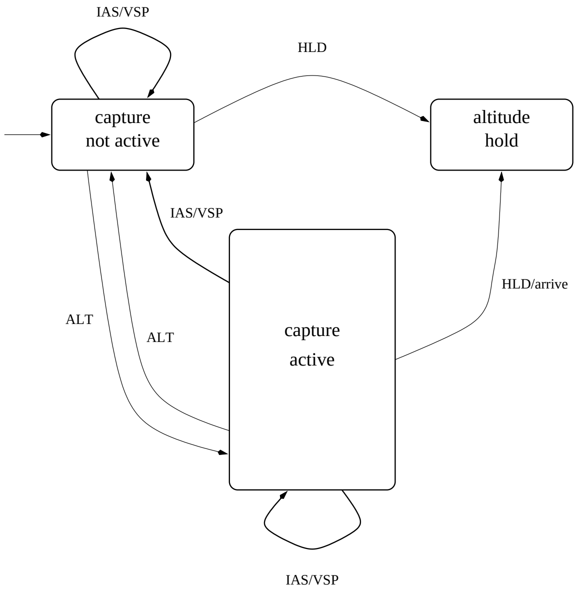

Figure 4: Abstracted State Machine for Actual System

`ALT`, `near`, `VSP` will lead to a surprise (capture will be active in the mental model but not in the actual system), and this is exactly the sequence that gave rise to the "kill the capture" scenario.

The diagrammatic approach used above is intended only to convey the intuition behind this method of analysis. To obtain a method suitable for practical application, we need the reliability and power of automated tools. Model checkers such as Murphi, SMV, and SPIN (which are all freely available) can provide this automation. To apply a model checker to these problems, we first specify the actual system and the mental model as state machines in the language of the tool concerned, then specify the expected abstraction relationship between the two descriptions, and cause the model checker to search for a violation of this relationship. Model checkers work by performing (explicit or symbolic) search over the entire state space of the state machines supplied to them; what makes them useful is that they can explore vast numbers of states (many millions) in a reasonable time. (Model checkers differ from simulators in that they consider all possible scenarios.) A companion paper [7] describes how the example considered here can be subjected to automated analysis using the model checker Murphi from David Dill's group at Stanford. The specification of the actual system and mental model requires just a few dozen lines of specification, and the automated analysis finds the "kill the capture" scenario in a fraction of a second. More interestingly, the analysis finds another surprise in a modified system specification that is intended to correct the source of the original surprise, and then another surprise in a further modification. The analysis was also extended to model the behavior of a faulty operator (who "misremembers" the current mode) and the utility of displays in correcting this.

Discussion

There is much excellent work in the fields of system design, aviation psychology, ergonomics and human factors that seeks to understand and reduce the sources of operator error in automated systems. The method described here is intended to complement these existing studies by providing a practical, mechanized means to examine system designs for features that may be error prone. Human factors and other studies provide an idea of what to look for, and the method described here provides a method to look for it. The method uses existing tools for model checking and state exploration that have, in other kinds of applications, scaled successfully to quite large systems.

Model checking is a member of the class of techniques known as "formal methods" and there has been prior work in applying formal methods to the problems of automation surprises. For example, Leveson and colleagues [5] compare system designs (by hand) against a list of design features that are prone to cause operator mode awareness errors. One of the error-prone design features identified by Leveson is use of “indirect” mode transitions—those which occur without explicit operator input. This approach was applied to the example considered here by Leveson and Palmer [4] and successfully identified the indirect pitch mode transition to `ALT CAP` in the example considered here, and the confusing interaction between the pitch and capture modes. However, that manual analysis did not report the surprise in the modified system description that was detected by our automated analysis using Murphi.

In other work that applies automated formal methods to mode confusion, Butler and colleagues [1] examine an autopilot specification for satisfaction of consistency and safety properties expressed as invariants.

We see the method presented here as complementary to these other approaches: automation is an adjunct, not a replacement, for careful human review of design specifications, and checking for consistent behavior is surely desirable in any analysis of an interactive system. What our method contributes is the ability to include an explicit model of the operator in the analysis. This allows detection of potential mode confusions that are beyond the scope of these other methods, but depends on the construction of suitable mental models.

For the future, we plan to apply this method to larger examples, and to evaluate its effectiveness in more realistic applications. We are also interested in using the technique to explore the consequences of operator error (and the effectiveness of remedies such as lock-ins and lockouts, or improved displays), and to assess the cognitive load placed on an operator by a given design (e.g., if the simplest mental model that can adequately track the actual system requires a large number of states, or a data structure such as a stack, then we may conclude that the design is too complex). We are also interested in exploring potential interactions between multiple mental models (e.g., for different aspects of the behavior of a single system), the consequences of inappropriate mental models (e.g., the interaction between a model of normal operation and a system operating in an unusual mode), and in using this method to assess training materials and checklists.

References

[1] Ricky W. Butler, Steven P. Miller, James N. Potts, and Victor A. Carreño. A formal methods approach to the analysis of mode confusion. In *17th AIAA/IEEE Digital Avionics Systems Conference*, Bellevue, WA, October 1998.

[2] Denis Javaux and Véronique De Keyser, editors. *Proceedings of the 3rd Workshop on Human Error, Safety, and System Development (HESSD’99)*, University of Liege, Belgium, June 1999.

[3] Denis Javaux and Peter G. Polson. A method for predicting errors when interacting with finite state machines. In Javaux and Keyser [2].

[4] Nancy G. Leveson and Everett Palmer. Designing automation to reduce operator errors. In *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, October 1997. Available at `http://www.cs.washington.edu/research/projects/safety/www/papers/smc.ps`.

[5] Nancy G. Leveson, L. Denise Pinnel, Sean David Sandys, Shuichi Koga, and Jon Damon Rees. Analyzing software specifications for mode confusion potential. In C. W. Johnson, editor, *Proceedings of a Workshop on Human Error and System Development*, pages 132–146, Glasgow, Scotland, March 1997. Paper available at `http://www.cs.washington.edu/research/projects/safety/www/papers/glasco%w.ps`.

[6] Everett Palmer. “Oops, it didn’t arm.” A case study of two automation surprises. In Richard S. Jensen and Lori A. Rakovan, editors, *Proceedings of the Eighth International Symposium on Aviation Psychology*, pages 227–232, The Aviation Psychology Laboratory, Department of Aerospace Engineering, Ohio State University, Columbus, OH, April 1995. Paper available at `http://olias.arc.nasa.gov/˜ev/OSU95_Oops/PalmerOops.html`.

[7] John Rushby. Using model checking to help discover mode confusions and other automation surprises. In Javaux and Keyser [2].

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.