Electronic Notes in Theoretical Computer Science 43 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume43.html` 14 pages

Analyzing Cockpit Interfaces Using Formal Methods

John Rushby[1]

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Abstract

Modern passenger aircraft are highly automated, and problems at the interface between the automation and the pilot are implicated in several accidents. I use a simple example taken from the autopilot of a widely used aircraft type to demonstrate how formal methods can be used to analyze some aspects of these interfaces, and to expose potential problems.

This example serves to illustrate the wider thesis that formal methods can find application in domains outside those traditionally associated with it, provided only that the phenomena of interest can be modeled effectively in discrete mathematics.

1 Introduction

The calculus was invented to solve problems of bodies in motion. In the few hundred years since Newton and Leibniz created its basic notations and techniques, the methods of calculus have been applied to a bewildering variety of problem areas from the motions of fluids to the behavior of biological systems. The reasons for this fecundity are well known: the fundamental concepts of calculus are very general and can be used to model many phenomena (indeed, students are usually introduced to it nowadays as the "science of change"), and it admits effective computational procedures that can be used to calculate answers to practical questions. The methods of calculation have been adapted to exploit the power of modern computers (indeed, the desire to automate these calculations was one of the spurs to the development of digital computers).

Formal methods are analogous to calculus in these regards: the basic concepts are very general (i.e., mathematical logic, which was itself developed to underpin the whole of mathematics—and which had its earliest roots in classical logic, which

[1] This research was partially supported by NASA Langley Research Center through contracts NAS1-20334 and NAS1-00079.

sought to codify the processes of rational thought) and can be used to model many phenomena, and they admit (what are beginning to be) effective computational procedures (such as model checking and theorem proving) that can be used to calculate answers to practical questions. These methods of calculation make good use of the power of modern computers.

I will illustrate these points with an example that applies formal methods to the "automation surprises" that have become a matter of concern in the cockpit automation of advanced aircraft (pilot error accounts for the majority of "hull loss" accidents in civil aviation [9]). This topic has traditionally been studied by aviation psychologists, cognitive psychologists, and human factors specialists using experimental and descriptive methods (see, e.g., [1]) and has not seemed a likely target for formal methods. It turns out, however, that cognitive psychologists have established the central role played by the "mental model" when a human interacts with any kind of complex system [13,14,15], and they have further established that these mental models have certain characteristics and can be described by state machines.

If certain aspects of human cognition can be described by state machines, then we can use formal methods to specify and model those aspects, and thereby provide effective means for calculating their properties. The next section describes a simple example to demonstrate this.

2 Analyzing an Automation Surprise

Automation surprises occur when an automated system behaves differently than its operator expects. If the actual system behavior and the operator's mental model are both described as finite state transition systems, then mechanized techniques based on model checking can be used automatically to discover any scenarios that cause the behaviors of the two descriptions to diverge from one another. These scenarios identify potential surprises and pinpoint areas where design changes, or revisions to training materials or procedures, should be considered. The mental models can be suggested by human factors experts, or can be derived from training materials, or can express simple requirements for "consistent" behavior.

I and my colleagues have previously demonstrated this approach by applying the Murϕ state exploration system to illustrative surprises in the autopilots of the MD88 [16,17] and A320 [4]. In this paper, I provide another illustration based on descriptions published by Degani and colleagues [5,6]. This illustration concerns the autopilot of one of the most widely deployed of all commercial aircraft types, and differs from previous examples in that it is necessary to model rather more of the dynamics of the aircraft.

Figure 1 illustrates the main elements of the Guidance Control Panel (GCP) that the pilot of this aircraft uses to select and control the vertical flight modes of the autopilot. There are three main modes, each of which is engaged by pressing the corresponding button.

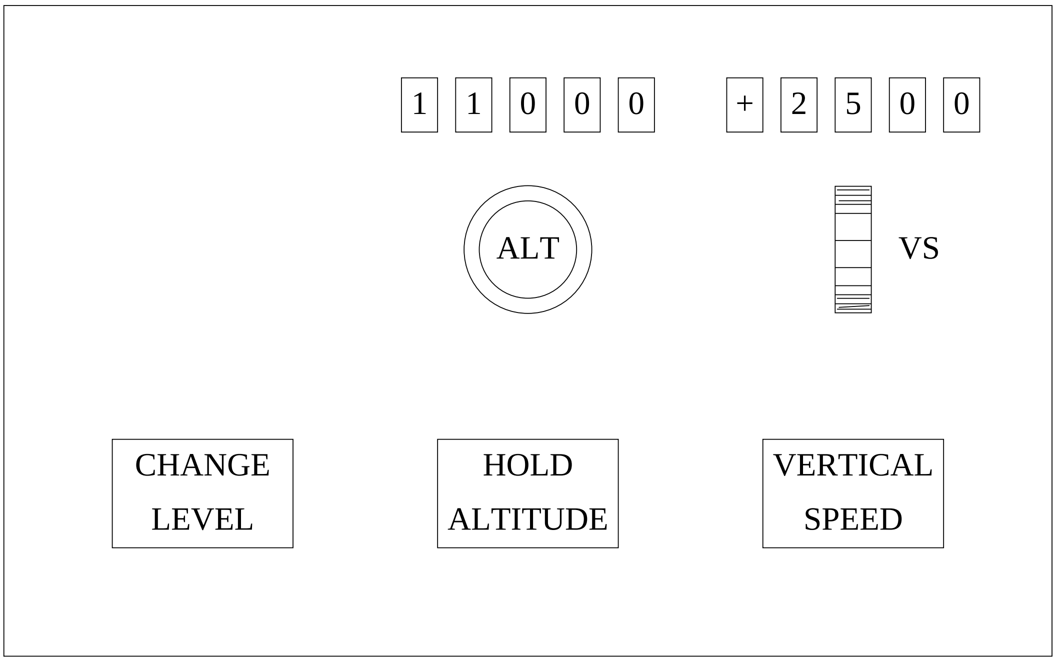

Fig. 1. The Guidance Control Panel

Hold Altitude: causes the aircraft to maintain its current altitude.

Change Level: causes the aircraft to fly to the new level indicated in the altitude window at the top center of the GCP. This value is set by turning the ALT dial in the center of the GCP. When the desired altitude is reached, the aircraft changes to *Hold Altitude* mode.

Vertical Speed: causes the aircraft to climb or descend at the rate indicated in the vertical speed window at the top right of the GCP (a positive value indicates climb, a negative one descent). This value can be changed by turning the VS thumbwheel at the center right of the GCP. If the value set in the altitude window is "ahead" of the current aircraft position (i.e., it is above the current altitude if the vertical speed window indicates climb, or below if it indicates descent), then there is a limit set and the aircraft will level off at the indicated altitude and enter *Hold Altitude* mode; otherwise, it will continue to climb or descend indefinitely at the set vertical speed.

The behaviors just described seem fairly simple. They become more complicated if the ALT dial or VS thumbwheel is moved when a mode is already active. For example, if the ALT dial is turned while in *Change Level* mode, the new altitude will become the target, whereas if the VS thumbwheel is turned, the aircraft will transition to *Vertical Speed* mode.

The behaviors of interest here are those that occur in *Vertical Speed* mode: if either the VS thumbwheel or ALT dial is changed when in this mode, then the aircraft will fly at the rate of climb or descent indicated in the vertical speed window, with a limit to the climb if the value in the altitude window is "ahead" of the current position. This behavior seems fairly straightforward but it is compromised by the existence of a special *Capture* mode that automatically becomes active when the aircraft gets close to the target altitude when a limit is set. The *Capture* mode causes the aircraft to make a gentle transition from climb or descent to level flight at the set altitude.

Before describing how this mode actually works, I describe some scenarios that it can exhibit. In each case, the scenario begins with the pilot selecting *Vertical Speed* mode with an ascent rate of 2,000 fpm and a limit of 27,000 feet; the aircraft will climb as directed and the autopilot will switch automatically to *Capture* mode as it passes through 25,000 feet. Now suppose that while passing through 26,000 feet, the pilot changes the ALT dial to a new value. Let us consider the behavior for different values set in the altitude window.

(i) If the new value is 27,500 feet, the aircraft will continue to ascend and will level off at this altitude. This seems reasonable because the new value is "ahead" of the aircraft and the one previously set.

(ii) If the new value is 26,500 feet, the aircraft will continue to ascend and will level off at this altitude. This also seems reasonable because, although the new value is lower than the one originally set, it is still "ahead" of the aircraft.

(iii) If the new value is 25,500 feet, the aircraft will change direction and descend to this altitude and level off. Whereas the previous two cases are consistent with behavior of *Vertical Speed* mode, this one is not: the new set altitude is "behind" the current position of the aircraft—so in *Vertical Speed* mode the aircraft would now climb without limit. How is a pilot to make sense of the new behavior? One plausible hypothesis is that *Capture* mode behaves somewhat like *Change Level* mode and always targets the set altitude, regardless of the setting in the vertical speed window.

The actual mechanism that produces these behaviors is rather different than the mental model hypothesized above. The autopilot notes the altitude at which it makes its automatic transition to *Capture* mode, and will capture any new height set in the altitude window that is "ahead" of that altitude; otherwise, it will change to *Vertical Speed* mode and climb or descend without limit.

Our goal is to determine whether the behavior suggested by the mental model can diverge from that of the actual mechanism. We will explore these behaviors by modeling them with the state exploration tool (a kind of model checker) called Murϕ [7]. I provided a tutorial introduction to use of Murϕ in this domain in an earlier paper [16] and therefore present the new model without much explanation of the Murϕ constructs used. The point to bear in mind is that a Murϕ specification consists of a number of guarded `rule`s (or `rulesets`): at each step, some rule whose guard evaluates to *true* in the current state is selected for execution. The body of the selected rule is executed, which will result in changes to the values of some state variables (and hence a new state). Murϕ backtracks to explore different sequences in which rules may fire and thereby examines all possible behaviors and all reachable states. In each reachable state, it checks whether specified invariants are *true*, and stops and prints a trace of the events (i.e., rule firings) that led to the current state whenever a violation is detected.

In this domain, rules specify actions by the pilot (e.g., pressing a button, or changing the value set by a dial), those corresponding to the dynamics of the aircraft (e.g., a change in the current altitude of the aircraft), and those performed by the autopilot in response to certain events (e.g., the change to *Capture* mode when the aircraft gets close to the target altitude). Unlike the exercise based on the MD-88 autopilot [16], where it was necessary to model only the discrete internal states of the autopilot, here we also need to model some of the dynamics of the aircraft—because its autopilot behaviors are determined by the relationships between the altitudes at which different events occur. However, a very crude model is sufficient: we abstract altitude to eleven discrete `flightlevels` (fewer would probably also be adequate) and allow the aircraft to move between adjacent flightlevels at each step of the model. We also need to record whether the aircraft is going `up` or `down`, and the current `vertical_mode` of its autopilot. These notions are specified in Murϕ as follows.

```
Const
  lo: 20;
  hi: 30;
Type
  flightlevels: lo..hi;
  directions: enum {up, down};
  vertical_modes: enum
    {none, hold_alt, vert_speed, change_level, capture};
```

To model the autopilot, we need state variables to record its current `flight_mode`, and whether there is a `limit_set` to its climb or descent. We also need to record the `current` flightlevel of the aircraft, the setting of its `alt_dial`, and the flightlevel at which it engaged *Capture* mode (this is the variable `cap_start`). The setting of the `vspd_wheel` can be abstracted to whether it indicates `up` or `down`, and we also need to record the current `direction` of the aircraft. The values of most of these state variables are presented to the pilot on a display called the *Flight Mode Annunciator* (FMA) and may therefore be assumed also to be present (and correctly represented) in the pilot's mental model. Conspicuously absent from the FMA, however, are `cap_start` (of whose existence pilots are unaware) and `limit_set`. The task of the mental model that we are investigating is to infer the value of `limit_set` from the other information presented. We record the value of this inference in the state variable `mental_capture`. These state variables are specified and initialized in Murϕ as follows.

```
Var
  flight_mode: vertical_modes;
  cap_start, current, alt_dial: flightlevels;
  direction, vspd_wheel: directions;
  limit_set: boolean;
  mental_capture: boolean;

startstate
begin
  flight_mode := none;
  current := (lo+hi)/2;
  clear mental_capture;
  clear cap_start;
  clear alt_dial;
  clear direction;
  clear vspd_wheel;
  clear limit_set;
end;
```

When the autopilot is disengaged (i.e., `flight_mode` is `none`), we assume the aircraft is being "hand flown" by the pilot and will move either up or down one flightlevel at each time step (taking care not to move beyond the allowed range of flightlevels).

```
ruleset d:directions do
rule "hand flight" flight_mode = none ==>
begin
  if d = up & current<hi then current := current+1;
  elsif d = down & current>lo then current := current-1;
  endif;
end;
end;
```

Otherwise, the aircraft is in "auto flight" and will move one flightlevel in its current direction, provided it is not in `hold_alt` mode, or already at the flightlevel set in its `alt_dial` (again taking care not to move beyond the allowed range of flightlevels).

```
rule "auto flight"
  (flight_mode != none & flight_mode != hold_alt)
    & current != alt_dial ==>
begin
  if direction = up & current<hi
    then current := current+1;
  elsif direction = down & current>lo
    then current := current-1;
  endif;
end;
```

The following rule specifies what happens when the pilot presses the *Vertical Speed* button.

```
rule "Engage vertical speed" flight_mode != vert_speed ==>
begin
  flight_mode := vert_speed;
  direction := vspd_wheel;
  limit_set := (direction = up & alt_dial >= current)
             | (direction = down & alt_dial <= current);
  mental_capture := limit_set
end;
```

The `direction` is set from the `vspd_wheel` (how this is set is described below), and `limit_set` is determined by whether the value set by the `alt_dial` (also described below) is "ahead" of the current flightlevel. The mental model is the same as the actual automation in this case, so `mental_capture` will be the same as `limit_set`.

The rules that specify behavior when the pilot presses the *Change Level* or *Hold Altitude* buttons are similar.

```
rule "Engage change level" flight_mode != change_level ==>
begin
  flight_mode := change_level;
  if alt_dial > current
    then direction := up; else direction := down;
  endif;
  limit_set := true;
  mental_capture := limit_set;
end;

rule "Engage hold alt"  flight_mode != hold_alt ==>
begin
  flight_mode := hold_alt;
  limit_set := false;
  mental_capture := limit_set;
end;
```

The transition into *Capture* mode from *Change Level* or *Vertical Speed* occurs autonomously when the aircraft comes "near" to the set altitude and there is a limit set. We do not need to model exactly what "near" means and simply allow this rule to fire whenever the basic constraints are satisfied.

```
rule "near"
  (flight_mode = change_level | flight_mode = vert_speed)
    & limit_set  ==>
begin
  flight_mode := capture;
  cap_start := current;
end;
```

Notice that the current flightlevel is noted in the `cap_start` state variable whenever this transition occurs.

The autopilot transitions automatically from *Capture* mode to *Hold Altitude* when it reaches the set altitude.

```
rule "arrive"
  flight_mode = capture & current = alt_dial ==>
begin
  flight_mode := hold_alt;
  limit_set := false;
  mental_capture := false;
end;
```

Next, we specify what happens when the pilot changes the ALT dial to a flightlevel `h`. If the current mode is *Vertical Speed*, then `limit_set` is set or not depending on whether the new value of the `alt_dial` is "ahead" of the `current` flightlevel; `mental_capture` is set the same way. If the autopilot is in *Capture* mode, however, then `limit_set` is is reevaluated depending on whether `alt_dial` is "ahead" of the `cap_start` flightlevel, whereas `mental_capture` is not changed.

```
ruleset h: flightlevels do
rule "change ALT dial" h != alt_dial ==>
begin
  alt_dial := h;
  if flight_mode = vert_speed then
    limit_set := (direction = up & alt_dial >= current)
               | (direction = down & alt_dial <= current);
    mental_capture := limit_set;
  elsif flight_mode = capture then
    limit_set := (direction = up & alt_dial >= cap_start)
               | (direction = down & alt_dial <= cap_start);
    flight_mode := vert_speed;
  elsif flight_mode = change_level then
    if h > current
       then direction := up; else direction := down;
    endif;
  endif;
end;
end;
```

Finally, we specify what happens when the pilot changes the vertical speed thumbwheel. If the current flight mode is *Vertical Speed*, or if it is *Hold Altitude* and the altitude setting has already been changed away from the current flightlevel, then the flight mode becomes *Vertical Speed* and `limit_set` is set or not depending on whether the value of the `alt_dial` is "ahead" of the `current` flightlevel.

```
ruleset d:directions do
rule "change VS thumbwheel" d != direction ==>
begin
  direction := d;
  if flight_mode = hold_alt & alt_dial != current then
    flight_mode := vert_speed;
    limit_set := (direction = up & alt_dial >= current)
               | (direction = down & alt_dial <= current);
    mental_capture := limit_set;
  elsif flight_mode = vert_speed then
    limit_set := (direction = up & alt_dial >= current)
               | (direction = down & alt_dial <= current);
    mental_capture := limit_set;
  endif;
end;
end;
```

Our goal is to check whether `mental_capture` always aligns with `limit_set`, so we specify the following invariant.

```
invariant "consistent"
  flight_mode != none -> mental_capture = limit_set;
```

If we now invoke Murϕ on this specification, it will explore a few hundred states in a fraction of a second and report that the invariant `consistent` can be falsified, and it will exhibit a trace such as that shown in Figure 2 that manifests this behavior. [2] This trace is similar to the scenarios described on page 4, except that the new value set in the ALT dial is "behind" that at which the automatic transition into *Capture* mode takes place—but the behavior of the aircraft is completely different in this case. Whereas the aircraft captured the new altitude in the previous cases, here it climbs without limit.

The insight provided by the Murϕ trace allows us to construct the following addition to the scenarios on page 4.

(iv) If the new value is 24,500 feet, the aircraft will climb without limit.

Observe how apparently small changes in inputs, namely the new value set in the altitude window, cause widely different behaviors, as tabulated below.

[2] Actually, Murϕ finds a much shorter counterexample trace than this; to simplify comparison with other scenarios, I disabled the rules `Engage change level` and `change VS thumbwheel`, and added the expression `current>cap_start & alt_dial> (lo+hi)/2` to the antecedent in `consistent`, thereby causing Murϕ to find this particular trace.

```
Startstate Startstate 0 fired.
flight_mode:none
cap_start:20
current:24
alt_dial:20
direction:up
vspd_wheel:up
limit_set:false
mental_capture:false
----------
Rule change_GCP_alt, h:27 fired.
alt_dial:27
----------
Rule Engage vert speed fired.
flight_mode:vert_speed
limit_set:true
mental_capture:true
----------
Rule auto flight fired.
current:26
----------
Rule auto flight fired.
current:27
----------
Rule near fired.
flight_mode:capture
cap_start:27
----------
Rule change_GCP_alt, h:26 fired.
The last state of the trace (in full) is:
flight_mode:vert_speed
cap_start:27
current:28
alt_dial:26
direction:up
vspd_wheel:up
limit_set:false
mental_capture:true
----------
End of the error trace.
```

Fig. 2. Counterexample Trace Found by Murϕ

Value in Altitude Window	Behavior
26,500	climb and capture
25,500	descend and capture
24,500	climb without limit

Our analysis with Murϕ has revealed a circumstance in which the hypothesized mental model differs from the behavior of the actual automation, and thereby highlights a potential automation surprise. This exact surprise was experienced by a crew in 1989. Degani and Heymann [5] provide the following incident report, paraphrased from [2].

> *"On climb to 27,000 feet and leaving 26,500 feet, Memphis Center gave us a clearance to descend to 24,000 feet. The aircraft had gone to '*Capture*' mode when the First Officer selected 24,000 feet on the GCP altitude setting... the aircraft continue to climb at approximately 300 feet-per-minute. There was no altitude warning and this 'altitude bust' went unnoticed by myself and the First Officer, due to the slight rate of climb. At 28,500 feet, Memphis Center asked our altitude and I replied 28,500 and started an immediate descent to 24,000 feet."*

An "altitude bust" is pilot jargon for the situation where an aircraft departs from its assigned altitude—potentially encroaching into airspace assigned to another aircraft and thereby causing a hazard.

We hypothesized the mental model that leads to this surprise by noting that in most common circumstances, the aircraft does capture the new altitude. Javaux and Polson [12] note that pilot's mental models focus on the common cases, so this model is psychologically plausible. However, pilots are *supposed* to operate the aircraft in accordance with the manufacturer's and their company's established procedures. I do not have access to descriptions of such procedures, but Degani et al. [6] present information from the training manual of the autopilot concerned. Part of the purpose of a training manual is to induce appropriate mental models, so it is worth examining this description.

The training manual indicates that changing the ALT dial when in *Capture* mode causes a reversion to *Vertical Speed* mode; implicitly, the usual rules for entering this mode will determine whether or not there will be a limit to the climb or descent (i.e., a limit will be set if the new value is "ahead" of the current altitude). It requires changing only a couple of lines in the Murϕ specification to implement this change to the operation of `mental_capture`. Murϕ then produces a counterexample trace similar to that of Scenario (iii) on page 4, showing that this mental model also admits surprises.

It is easy to establish that the model induced by the training manual leads to surprises only of the kind where the aircraft flies a capture when the mental model expects an unconstrained climb or descent, whereas the original "folk model" leads only to surprises of the opposite kind. Is it possible that some "compromise" model could eliminate these surprises altogether?

Unfortunately, this is not possible. Certain behaviors of the autopilot are determined by the relationship between a new value set in the altitude window and the altitude at which *Capture* mode was entered. This latter information (i.e., the `cap_start` altitude) is simply not presented to the pilot[3] and it is therefore impossible to construct a mental model whose states are based on information available to the pilot that accurately tracks the behavior of the real autopilot.

My opinion (shared with others who have considered this example [5,6]), is that the behavior of this autopilot is unacceptable, and it should be considered a flawed design. An automated system may have complex internal logic, and that logic may depend on state variables that are not displayed at its user interface, but its operation should be such that a relatively simple mental model can accurately predict the consequences of actions by its user. The victims of the automation surprise reported on page 11 rightly attributed it to a design flaw and suggested a correction (paraphrased from [2]).

> *"This problem could have been alleviated during certification, if the FAA had required the manufacturer to put the logic into the computer so as not to allow the aircraft to climb above the last assigned altitude when a lower one is selected before reaching your cruise altitude. This is a design flaw, and should be corrected."*

The suggested correction would avoid the altitude bust but would not, in my opinion, support a simple and accurate mental model. Corrections that would support such a model include a light by the altitude window that is illuminated if a capture will occur (i.e., if `limit_set` is true), or a more radical redesign in which turning the ALT dial when in *Capture* mode causes a reversion to *Level Change* (rather than *Vertical Speed*) mode.

3 Discussion

A limitation to the method of analysis presented here is that we are modeling only a small fragment of the cognitive processes involved in human-computer interaction. The method is silent, for example, on problems that might be due to an operator's difficulty in recalling the right mental model, or to excessive demands on an operator's attention. There is very interesting work by Bowman and Faconti [3] and by Duke and Duce [8] that applies formal methods to deeper models of cognition, and this allows them to detect different kinds of issues than the automation surprises described here. I consider all these approaches complementary to each other and representative of a very promising general direction: the detection of potential human factors problems by explicitly comparing the design of a system against a model of some aspect of human cognition using mechanized formal methods. Models of different aspects of cognition are likely to reveal different kinds of prob-

[3] A pilot could monitor the FMA for the transition to *Capture* mode and then note the altimeter reading, but this seems feasible only in theory, and is not consistent with the real responsibilities of flying an aircraft, nor with the intent of cockpit automation.

lems. The approach described here uses simple mental models to find design flaws that lead to automation surprises, and it seems very effective for that purpose.

Murϕ was developed to support modeling and analysis of asynchronous communication protocols, and its main applications have been processor cache coherence protocols (see the Murϕ home page at `http://verify.stanford.edu/dill/murphi.html`). Yet, I have found it quite convenient for modeling autopilots and their mental models. In my opinion, the key to successful use of formal methods "elsewhere" than to their traditional applications of hardware and protocol designs is use of effective tools that are not overly committed to a particular mode of expression or model of computation. Murϕ, with its simple guarded rules and its convenient notation for programming their bodies, has proved effective in domains quite far from its original applications (e.g., my colleagues and I have also used it to gain quantitative estimates of the fault tolerance of different algorithms [10]). Research in many traditional sciences, such as physics, chemistry, biology—and even psychology, recently has come to be based on simulation and computation over mathematical models. I believe that formal methods, by providing effective methods for enumeration of finite models, and symbolic exploration of infinite ones, will become a key technology in these sciences, and in many other fields of endeavor far "elsewhere" from its origins.

References

[1] Kathy Abbott, Jean-Jacques Speyer, and Guy Boy, editors. *International Conference on Human-Computer Interaction in Aeronautics: HCI-Aero 2000*, Toulouse, France, September 2000. Cépaduès-Éditions.

[2] ASRS report 113722. NASA Aviation Safety Reporting System (ASRS), June 1989. ASRS database search is available at `http://nasdac.faa.gov/asp/asy_asrs.asp`.

[3] Howard Bowman and Giorgio Faconti. Analysing cognitive behaviour using LOTOS and Mexitl. *Formal Aspects of Computing*, 11(2):132–159, 1999.

[4] Judith Crow, Denis Javaux, and John Rushby. Models and mechanized methods that integrate human factors into automation design. In Abbott et al. [1], pages 163–168. `http://www.csl.sri.com/reports/html/hci-aero00.html`.

[5] Asaf Degani and Michael Heymann. Pilot-autopilot interaction: A formal perspective. In Abbott et al. [1], pages 157–168.

[6] Asaf Degani, Michael Heymann, George Meyer, and Michael Shafto. Some formal aspects of human-computer interaction. Technical Report NASA/TM–2000–209600, NASA Ames Research Center, Moffett Field, CA, April 2000.

[7] David L. Dill. The Murϕ verification system. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[8] David Duke and David Duce. The formalization of a cognitive architecture and its application to reasoning about human computer interaction. *Formal Aspects of Computing*, 11(6):665–689, 1999.

[9] The interfaces between flightcrews and modern flight deck systems. Report of the FAA human factors team, Federal Aviation Administration, 1995. Available at `http://www.faa.gov/avr/afs/interfac.pdf`.

[10] Li Gong, Patrick Lincoln, and John Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In Ravishankar K. Iyer, Michele Morganti, W. Kent Fuchs, and Virgil Gligor, editors, *Dependable Computing for Critical Applications—5*, volume 10 of *Dependable Computing and Fault Tolerant Systems*, pages 139–157, Champaign, IL, September 1995. IEEE Computer Society.

[11] 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.

[12] Denis Javaux and Peter G. Polson. A method for predicting errors when interacting with finite state machines. In Javaux and Keyser [11]. Also to appear in *Reliability Engineering and System Safety*.

[13] Philip N. Johnson-Laird. *Mental Models*, volume 6 of *Cognitive Science Series*. Harvard University Press, Cambridge MA, 1983.

[14] Philip N. Johnson-Laird. *The Computer and the Mind : An Introduction to Cognitive Science*. Harvard University Press, 1989.

[15] Steven Pinker. *How the Mind Works*. W. W. Norton, New York, NY, 1997.

[16] John Rushby. Using model checking to help discover mode confusions and other automation surprises. In Javaux and Keyser [11]. Revised version to appear in *Reliability Engineering and System Safety*; Preprint available at `http://www.csl.sri.com/papers/hessd99/`.

[17] John Rushby, Judith Crow, and Everett Palmer. An automated method to detect potential mode confusions. In *18th AIAA/IEEE Digital Avionics Systems Conference*, St Louis, MO, October 1999. `http://www.csl.sri.com/papers/dasc99/`.