A FORMAL METHODS APPROACH TO THE ANALYSIS OF MODE CONFUSION

Ricky W. Butler, NASA Langley Research Center, Hampton, Virginia
Steven P. Miller, Rockwell Collins, Cedar Rapids, Iowa
James N. Potts, Rockwell Collins, Cedar Rapids, Iowa
Victor A. Carreno, NASA Langley Research Center, Hampton, Virginia

Introduction

The goal of the new NASA Aviation Safety Program (AvSP) is to reduce the civil aviation fatal accident rate by 80% in ten years and 90% in twenty years. This program is being driven by the accident data with a focus on the most recent history. Pilot error is the most commonly cited cause for fatal accidents (up to 70%) and obviously must be given major consideration in this program. While the greatest source of pilot error is the loss of "situation awareness", mode confusion is increasingly becoming a major contributor as well. The January 30, 1995 issue of Aviation Week lists 184 incidents and accidents involving mode awareness including the Bangalore A320 crash 2/14/90, the Strasbourg A320 crash 1/20/92, the Mulhouse-Habsheim A320 crash 6/26/88, and the Toulouse A330 crash 6/30/94 [2].

These incidents and accidents reveal that pilots sometimes become confused about what the cockpit automation is doing. Consequently, human factors research is an obvious investment area. However, even a cursory look at the accident data reveals that the mode confusion problem is much deeper than just training deficiencies and a lack of human-oriented design. This is readily acknowledged by human factors experts. For example, Charles E. Billings, writes in Aviation Automation: The Search for a Human-Centered Approach (pg 144) [1]:

> *…today's flight management systems are "mode rich" and it is often difficult for pilots to keep track of them ... The second problem, which is related to the first involves lack of understanding by pilots of the system's internal architecture and logic, and therefore a lack of understanding of what the machine is doing, and why, and what it is going to do next.*

Similarly, Sarter and Woods write [7]:

> *What is needed is a better understanding of how the machine operates, not just how to operate the machine.*

It seems that further progress in human factors must come through a deeper scrutiny of the internals of the automation. It is in this arena that formal methods can contribute. Formal methods refers to the use of techniques from logic and discrete mathematics in the specification, design, and verification of computer systems, both hardware and software. The fundamental goal of formal methods is to capture requirements, designs and implementations in a mathematically based model that can be analyzed in a rigorous manner. Research in formal methods is aimed at automating this analysis as much as possible. By capturing the internal behavior of a flight deck in a rigorous and detailed formal model, the dark corners of a design can be analyzed.

This paper will explore how formal models and analyses can be used to help eliminate mode confusion from flight deck designs and at the same time increase our confidence in the safety of the implementation. The paper is based upon interim results from a new project involving NASA Langley and Rockwell Collins in applying formal methods to a realistic business jet Flight Guidance System (FGS).

The Targeted Flight Guidance System

A *Flight Guidance System* (FGS) is a component of the overall *Flight Control System* (FCS) (see Figure 1). The FGS compares the measured state of an aircraft (position, speed, and attitude) to the desired state and generates pitch and roll guidance commands to minimize the difference between the measured and desired state. When engaged, the *Autopilot* (AP) translates these commands into movement of the aircraft's control surfaces necessary to achieve the commanded changes about the lateral and vertical axes. An FGS can be further broken down into the *mode logic* and the *flight control laws*. The mode logic accepts commands from the flight crew, the *Flight Management System* (FMS), and information about the current state of the aircraft to determine which system modes are *active*. The active modes in turn determine which flight control laws are used to generate the pitch and roll guidance commands. The active lateral and vertical modes are displayed (annunciated) to the flight crew on the *Flight Director* (FD), a portion of the Electronic Flight Instrumentation System (EFIS). The magnitude and direction of the lateral (roll) and vertical (pitch) commands generated by the FGS are also displayed on the EFIS as *guidance cues*.

The specification of the Flight Guidance System used in this project was developed at Collins as part of a project to investigate different methods of modeling requirements [5]. While it is a simplified composite of several actual Flight Guidance Systems, and does not describe an actual aircraft in service, it is complex enough to serve as a realistic example [4].

Goals of Formal Modeling

Moving new technology into practice is always more difficult than the creation of that technology, especially in the case of software development. Over the years a multitude of new software development methodologies have been produced, yet few of them have been accepted

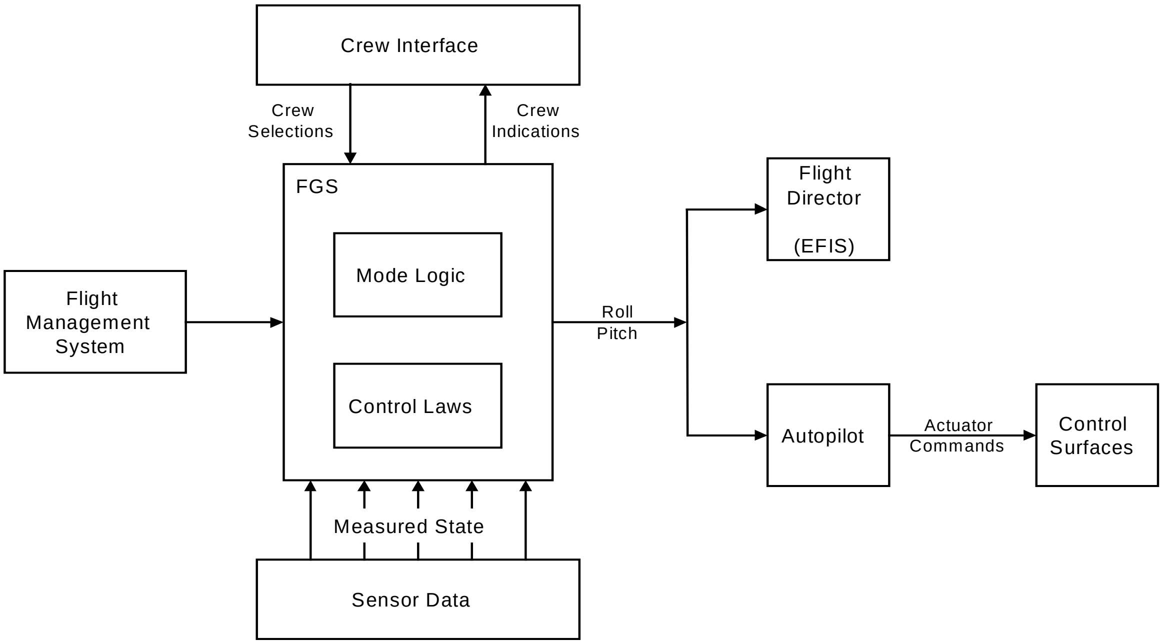

Figure 1 - Flight Control System Overview

by industry. A major goal of this project was not only to show how formal methods could be used to remove mode confusion from a flight guidance system and to discover design flaws, but to do so in a way that would be accepted by industry.

Companies such as Rockwell Collins typically build variations of the same products over and over. Increasingly, these companies are looking for strategies that support the systematic reuse of common artifacts. One such approach is Product Family Engineering, also known as Domain Engineering. Central to the Product Family approach is the development of a domain architecture consisting of those requirements, design, implementation, and verification artifacts that are common to all members of the family and the variations of these artifacts that are supported by the domain. Prior to this project, Collins had conducted a Commonality Analysis [9] of the FGS mode logic described in [4] and developed a tentative product family architecture. Consequently, an important goal of this project was to build on that work and develop a formal model consistent with that architecture.

Another central goal of the project was to make the mode logic accessible to pilots and experts in human factors. To achieve this, the mode logic is also specified as an executable ObjecTime model [8]. This model is connected to a mock-up of the Flight Deck so that the model can be executed by pressing buttons and turning dials on the mock-up. A visualization of the mode logic (Figure 3) is also displayed as the model executes, allowing pilots, experts in human factors, and the design engineers to relate the behavior of the automation to the human computer interface.

Yet another goal was to be able to formally analyze the model for various forms of consistency, completeness, safety properties, and properties related to human factors. To achieve this, the ObjecTime model was manually translated into the PVS specification language [6]. The desired properties could then be analyzed with this model using the PVS theorem prover. This overall strategy is illustrated in Figure 2.

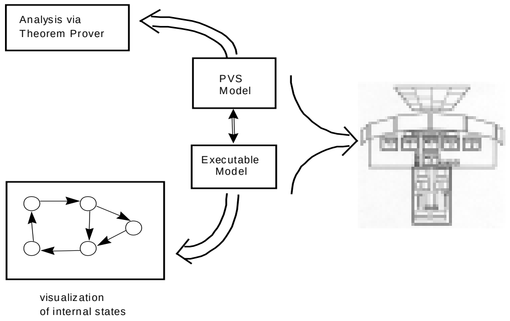

Figure 2 - Overall Strategy

As a result, there were several goals that affected the style of the formal models. In particular, the formal models had to be:

1. suitable for mathematical analysis of their potential for mode confusion
2. consistent with the executable ObjecTime model
3. conceptually simple enough to display during pilot training
4. consistent with the product family architecture

Interestingly, we found that (1) and (4) worked against each other. The proofs became more difficult as the model was structured to achieve (4). This will be discussed in detail in a future report.

Mode Confusion

Mode confusion can be traced to at least three fundamental sources: (1) opacity (i.e., poor display of automation state), (2) complexity (i.e., unnecessarily complex automation), and (3) incorrect mental model (i.e., flight crew misunderstands the behavior of the automation). Traditional human factors research has concentrated on (1), and significant progress has been made. However, mitigation of mode confusion will require addressing problem sources (2) and (3) as well. Towards this end, our approach uses two complementary strategies based upon a formal model:

Visualization Create a clear, executable formal model of the automation that is easily understood by flight crew and use it to drive a flight deck mockup from the formal model.

Analysis Conduct mathematical analysis of the model.

It is hoped that this approach will (1) force designers to commit to a clear conceptual model of the automation, (2) facilitate discussion between designers, human factors experts, and the flight crew, (3) enhance the training process by direct exposure to an accurate mental model of the automation, and (4) through analysis, uncover characteristics of the automation that historically have been a source of mode confusion.

Model Visualization

Development of a flight deck around an executable formal model enables several innovative strategies for pilot training. First, the executable model can be used in place of a rapid prototype for early life cycle discussions with pilots. These discussions can be focused on the

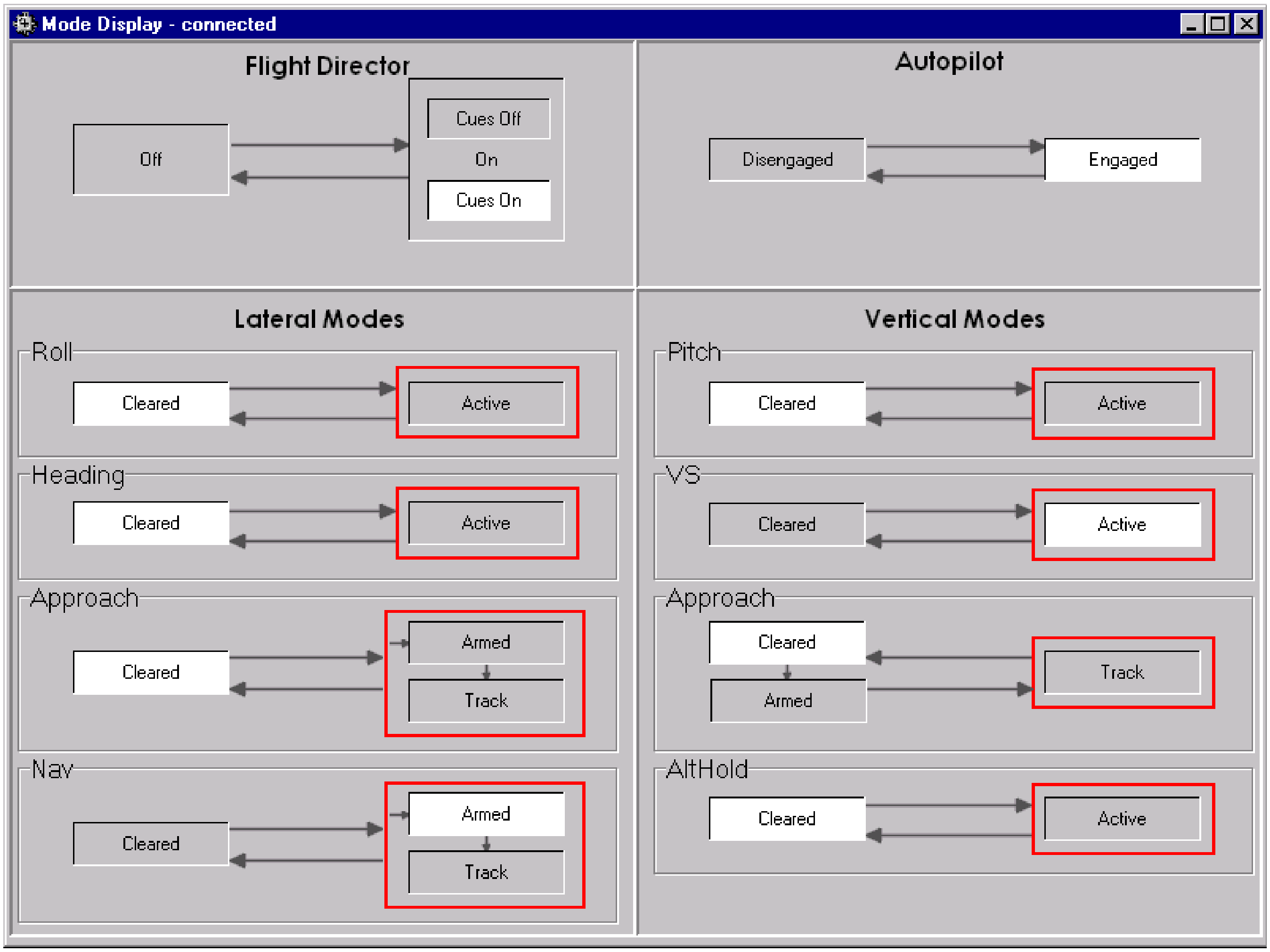

Figure 3 - Visualization of the FGS Modes

identification of model complexities that are confusing to the pilots. Second, during training the flight deck can be augmented with an additional display that directly exposes the internal structure of the automation and its dynamic changes. This display will not be present in the cockpit of an operational aircraft. It will be used exclusively to help the pilot form an accurate model of what the automation is doing.

An example of the visual display of the mode logic is shown in Figure 3. The state of the Flight Director (FD), Autopilot (AP), and each of the lateral and vertical modes are modeled as small, tightly synchronized finite state machines. In Figure 3, the FD is *On* with the guidance cues displayed; the AP is *Engaged*; lateral Roll, Heading, and Approach modes are *Cleared*; lateral NAV mode is *Armed*; vertical modes Pitch, Approach, and AltHold are *Cleared*; and the VS mode is *Active*. Active modes are those that actually control the aircraft when the AP is engaged. These are indicated by the heavy dark boxes around the *Active*, *Track*, and lateral *Armed* modes.

A small number of constraints govern most of the synchronization between the mode machines. For example, if the Flight Director is turned on (so that the lateral and vertical modes are annunciated on the Flight Director), then one, and only one, lateral mode can be active. A similar constraint holds for the vertical modes. Since the lateral and vertical modes are so tightly synchronized, a common mistake when modeling the mode logic is to try to combine the lateral and vertical mode machines into a single lateral mode machine and a single vertical mode machine. Besides violating the modularity needed to support a family of products, combining the modes in this way breaks down when other modes that are more loosely synchronized are added.

This visual model of the automation is connected to a simulation of the cockpit and can be executed by pressing buttons and turning dials on the mockup. In this way, pilots, experts in human factors, and the designers can easily relate the behavior of the automation with the human computer interface

Model Analysis

In a new paper entitled "Analyzing Software Specifications for Mode Confusion Potential" [3], Nancy Leveson, et. al., identify six categories of design that have historically been a source of mode confusion: (1) inputs interpreted differently in different modes, (2) indirect mode changes, (3) behavior that is different in different modes, (4) operator authority limits, (5) unintended sides effects, (6) lack of appropriate feedback. The critical question here is whether these categories can be understood well enough that they can be mathematically characterized. If so, the formal models can be analyzed against these characterizations. Although this work is incomplete, we can illustrate the concept on categories (2) and (3) above. The analysis is made possible by the translation of the ObjectTime visualization model into the PVS specification language. This was done manually for this project, but future work will look into automatic translation.[1]

The formal PVS specification is centered around a "next_state" function that defines the overall system transition in terms of several synchronized state machines. The system state vector includes four fields: LATERAL, VERTICAL, FD, and AP which contain the state of the lateral guidance, vertical guidance, Flight Director and Autopilot. The lateral and vertical guidance models are further defined in terms of several synchronized mode machines such as PITCH, ROLL, NAV, and HDG. Details of this model will be included in a future technical report.

[1] Alternatively one could drive the visualization from the PVS specification. The ultimate goal is to use one model for specification, training, analysis, and implementation.

Indirect Mode Changes

The first problem is formally defining what constitutes an indirect mode change. Let's begin by defining it as a mode change that occurs when there has been no crew input:

```
Indirect_Mode_Change?(s,e): bool =
   NOT Crew_input?(e) AND Mode_Change?(s,e)

No_Indirect_Mode_Change: LEMMA
    Valid_State?(s) IMPLIES
          NOT Indirect_Mode_Change?(s,e)
```

We then seek to prove the "false" lemma above using GRIND, a brute force proof strategy that works well on lemmas that do not involve quantification.[2] The resulting unproved sequents elaborate the conditions where indirect mode changes occur. For example,

```
{-1}    Overspeed_Event?(e!1)
{-2}    OFF?(mode(FD(s!1)))
{-3}    s!1  WITH [FD := FD(s!1) WITH [mode := CUES],
            LATERAL := LATERAL(s!1) WITH
                        [ROLL := (# mode := ACTIVE #)],
            VERTICAL := VERTICAL(s!1) WITH
                         [PITCH := (# mode := ACTIVE #)]]
        = NS
{-4}    Valid_State(s!1)
   |-------
{1}     mode(PITCH(VERTICAL(s!1))) =
        mode(PITCH(VERTICAL(NS)))
```

The situations where indirect mode changes occur are clear from the negatively labeled formulas in each sequent. We see that an indirect mode change occurs when the overspeed event occurs and the Flight Director is off. This event turns on the Flight Director and places the system into modes ROLL and PITCH.[3]

Discovering Inconsistent Behavior

Precisely defining the concept of inconsistent behavior is nontrivial and likely to be a long term endeavor. With no pretense of fully capturing the notion of inconsistent behavior, we offer the following as simple examples to illustrate the concept:

- Button pushes that are ignored in some modes but not others
- Button pushes that act like toggles in some modes but not others

We define an "ignored command" as one in which there is a crew input and there is no mode change. We seek to prove that this never happens:

```
No_Ignored_Crew_Inputs: LEMMA
    Valid_State(s) AND Crew_Input?(e) IMPLIES
        NOT Mode_Change?(s,e)
```

The result of the failed proof attempt is a set of sequents similar to the following:

```
{-1} VS_Pitch_Wheel_Changed?(e!1)
{-2} CUES?(mode(FD(s!1)))
{-3} TRACK?(mode(NAV(LATERAL(s!1))))
{-4} ACTIVE?(mode(VS(VERTICAL(s!1))))
|-------
{1} ACTIVE?(mode(ROLL(LATERAL(s!1))))
{2} ACTIVE?(mode(HDG(LATERAL(s!1))))
```

The negatively labeled formulas in the sequent clearly elaborate the case where an input is ignored, i.e., when the VS/Pitch Wheel is changed and the Flight Director is displaying CUES and the active lateral mode is ROLL and the active vertical mode is PITCH. In this way, PVS is used to perform a state exploration to discover all conditions where the lemma is false, i.e., all situations in which a crew input is ignored.

We can determine whether the HDG Switch acts like a toggle by seeking to prove the following lemma which asserts that the HDG mode toggles between CLEARED and ACTIVE whenever the HDG switch is pressed.

```
HDG_Toggle?: LEMMA
   HDG(LATERAL(next_state(s, HDG_Switch_Hit))) /=
      HDG(LATERAL(s))
```

This lemma is easily proved for the current model. Of course, this may not remain true as

[2] As the model has grown larger, we have had to develop more effective custom proof strategies to keep the proof times at manageable durations.

[3] PITCH mode is selected because the model is still under construction. In the final model, Flight Level Change (FLC) mode will be selected as the active vertical mode when overspeed occurs.

more modes are added and the mode logic becomes more complex.

Safety Analysis

Of particular importance to the analysis of safety-critical systems is the fact that formal methods provides a way to investigate all of the behaviors of a model. In other words, they can explore whether a property is true over its entire input space. And total exploration of the entire input space is the only way to gain assurance that catastrophic failure does not lie hidden among the vast number of possible behaviors. Simulation and testing simply cannot accomplish this in practical amounts of time. Although no accident in civil aviation has been blamed directly on a bug in the software, many serious incidents have occurred and are occurring with increasing frequency. Billings writes (pg. 149) [1]: *It must be noted that automation also makes apparently random, unpredictable "errors" (e.g. the flap lockup at Hong Kong, 1994)*. He then offers six examples where this has occurred.

A key advantage of a formal model is that it can be mathematically analyzed to insure that key safety properties are not violated. For example, one would seek to prove

```
GO_AROUND_SAFETY_PROP: LEMMA
    Valid_state(s) IMPLIES
        LATERAL(next_state(s,e) = GA    IFF
        VERTICAL(next_state(s,e) = GA
```

In other words, the lateral mode will only be in Go Around if the vertical mode is in Go Around and vice versa for all possible inputs and reachable states. This lemma is proved for all possible inputs, so conceptually this is equivalent to exhaustive testing[4]. Another example is:

```
At_Least_One_Lateral_Mode_Active(s): bool =
    ON?(FD(s)) IFF
        At_Least_One_Mode_Active(LATERAL(s))

ALOLMA: LEMMA
At_Least_One_Lateral_Mode_Active(s) IMPLIES
At_Least_One_Lateral_Mode_Active(next_state(s,e))
```

This property must be proved as a consequence of modeling the lateral modes as a set of parallel state machines.

Currently we have identified over 50 key functional properties. Three are shown here:

```
AP_TURNS_ON: LEMMA
    NOT Engaged?(AP(s))  AND
    Engaged?(AP(next_state(s, e)))
        IMPLIES
            AP_Engage_Switch_Pressed?(e)  AND
            Disconnect_Bar_Up?(AP(s))


FD_TURNS_OFF: LEMMA
    On?(FD(s))  AND
    NOT On?(FD(fast_next_state(s, e)))
        IMPLIES
            FD_Switch_Hit?(e) AND
            NOT Engaged?(AP(s)) AND
            NOT Overspeed?(FD(s))

HDG_SELECTED: LEMMA
    NOT ACTIVE?(mode(HDG(LATERAL(s))))
    AND HDG_Switch_Hit?(e)
        IMPLIES
ACTIVE?(mode(HDG(LATERAL(next_state(s,e)))))
```

It is interesting that the need to prove certain key properties has led us to discover even more key functional properties than had been cited in [4] and [5]. This has led us to speculate about the possibility of a more abstract specification in which these properties are explicitly stated, rather than being implicit as in the current operational model.

Future Work

Finally, because the current flight deck has grown incrementally over two decades, the result has not been a coherent, well integrated means of controlling an aircraft. Tony Lambregts, the FAA National Resource Specialist for Automated Controls writes "... the great majority of automation deficiencies and unnecessary complexities are the result of bad formulation of requirements and adhering too long to outdated

[4] Note. This is not a claim for perfection because we have no guarantee that all of the needed properties have been elaborated and that they have all been stated correctly.

technologies, engineering design concepts and processes." In particular he argues that much of the complexity of the flight deck derives from the independent design of the autopilot and autothrottle. Thus, it seems essential that a multidisciplinary approach using (1) integrated control laws (i.e. combining autopilot and autothrottle), (2) formal models, and (3) human factors will be necessary to fully solve the mode confusion problem. It is also clear that the formal models must be extended to include information about the control laws in order to enable a full analysis of systems that would result from such a multidisciplinary development. The crash of an Airbus A330-322 in Toulouse, France on 6/30/1994, highlights this need. During a test flight of simulated engine failure, an unexpected mode transition to altitude acquisition (ALT*) occurred. Pitch protection was not provided in ALT* mode, although it was present in all of the other modes. Detection of inconsistent behavior such as this will necessitate the elaboration of the basic properties of the control laws in the model in addition to the mode structure.

Conclusions

Many of the proposed solutions to other accident categories in the NASA AvSP program involve the use of new automation. It seems likely that as failures in other accident categories are reduced, the problems associated with automation will increase in significance. Use of formal methods can

- aid the pilot in training through direct display of internal states
- reveal "dark corners" and non-intuitive interactions through analysis
- discover properties of the design that have a potential for mode confusion
- discover design errors early in life cycle

References

[1] Charles E. Billings. Aviation Automation: The Search for a Human Centered Approach. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 1997.

[2] Dan Hughes and Michael Dornheim, Automated Cockpits: Who's in Charge?, Aviation Week & Space Technology, January 30-February 6, 1995

[3] Nancy Leveson, et al, Analyzing Software Specifications for Mode Confusion Potential, 1997.

[4] Steven P. Miller and Karl F. Hoech, Specifying the mode logic of a flight guidance system in CoRE, Rockwell Technical Report WP97-2011, Rockwell Collins, November 1997.

[5] Steven P. Miller, Specifying the Mode Logic of a Flight Guidance System in CoRE and SCR, in Proceedings of the Second Workshop on Formal Methods in Software Practice (FMSP'98), pg. 44-53, Clearwater Beach, Florida, March 4-5, 1998.

[6] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal Verification for Fault-tolerant Architectures: Prolegomena to the Design of PVS. IEEE Transactions on Software Engineering, 21(2):107-125, Feb. 1995.

[7] N.B. Sarter and D.D. Woods. Decomposing Automation: Autonomy, Authority, Observability and Perceived Animacy. First Automation Technology and Human Performance Conference, April 1994.

[8] Bran Selic, G. Gullekson, and P. Ward, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, 1994.

[9] David M. Weiss, Defining Families: The Commonality Analysis, Lucent Technologies Bell Laboratories, 1000 E. Warrenville Rd, Napierville, IL, 60566, 1997.