IDENTIFYING MODE CONFUSION POTENTIAL IN SOFTWARE DESIGN

by

Mario A. Rodríguez

B.S. Industrial Engineering
University of Central Florida, 1998.

Submitted to the Department of Aeronautics and Astronautics in Partial
Fulfillment of the Requirements for the Degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2000
[September 2000]

Author ....
Department of Aeronautics and Astronautics
July 28, 2000.

Certified by ... .
Professor Nancy Leveson
Department of Aeronautics and Astronautics
Thesis Supervisor

Accepted by....................... ....
Professor Nesbitt W. Hagood
Associate Professor of Aeronautics and Astronautics
Chairman, Department Graduate Committee

IDENTIFYING MODE CONFUSION POTENTIAL IN SOFTWARE DESIGN

by

Mario A. Rodríguez

Submitted to the Department of Aeronautics and Astronautics
on July 28, 2000, in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

Abstract

The current challenge of modern aircraft design is not in the capacity to automate functions, but in making automation and humans effective team players. There are many human factors issues that need to be considered in order to optimally operate the system in a safe manner. This is not an easy task. Initial attempts mitigating the occurrence of some known error types have produced new types of errors. Many of these new error forms are the result of what has been labeled *technology-centered automation,* which occurs when too little attention during software development is devoted to providing enough support to humans in their monitoring and control tasks. In the case of auto flight systems, automation has become a potential safety liability and there is a need to systematically detect these types of error forms.

In order to evaluate and examine automation features that could lead to mode confusion errors in complex process control systems, a formal specification language was used (SpecTRM-RL). The MD-11 vertical flight control system was chosen as a case study because of many reasons. First it was a very large and complex system and one of the contributions of this thesis was to evaluate the use of SpecTRM-RL for building complex systems. Elements of the model are provided as well as lessons learned from the effort of building the system. Another reason for choosing the vertical flight control system is because it has been reported as one of the most common sources of mode confusion errors in the ASRS reports. Finally, SpecTRM-RL is suited for the analysis of mode confusion. Ideas are presented over how future design criteria can be added to the existing set of tools available in SpecTRM-RL.

A human task visualization language is presented in order to build a model of the user tasks that are analyzable and translatable to SpecTRM-RL. A translation of the elements of the visual language is presented as well as suggestion on how it can be used for detecting three types design features that can lead to mode confusion.

Thesis Supervisor: Nancy G. Leveson
Title: Professor of Aeronautics and Astronautics

Dedication

Pra meu papazinho.

Table of Contents

List of Figures

1 Introduction

New advances in technology have added new functionality into systems. One of the clearest examples is the aviation industry where advances in computing have had a significant impact on the way systems are built. Some of the success achieved can be seen in savings in production time, reduction of workload, increased productivity, and increased precision. Unfortunately, humans have had problems coping with their automated counterparts, and this affects negatively the safety of the new automated systems. Such technology-centered automation has brought along surprises and new types of errors. Breakdowns in the interaction of users with automated systems are becoming more common. Pilots are at times lost in the automation. They ask questions like "What is the computer doing?" "Why is it doing that?" "What will it do next?" “How do I stop it from doing that?" and "How do I get it to do what I want it to do?" (Wiener, 1980). The resulting confusion, sometimes called mode confusion, has been cited as a contributing cause of accidents and serious incidents, highlighting the need to develop ways to prevent it.

These surprises are not limited to operators. Designers as well have experienced unexpected consequences (Sarter and Woods, 1995). As a result, a systematic methodology for detecting these error forms should be developed for designers, as well as for operators of these systems. It is hypothesized that such a methodology will allow organizing ideas more clearly and visualizing the behavior of these models in a more complete and systematic way, thus improving the overall safety of the system.

There are many challenges in improving the safety of a complex system. As technology continues to improve, it is easy to make systems more complex and designing them in a form that can be handled by humans is not an easy task. A common engineering approach in dealing with complexity is the use of modes. Modes are defined as mutually exclusive sets of system behaviors (Leveson, 1997). The use of modes allows for flexibility and provides operators with a set of tools capable of optimizing system performance across a wide range of circumstances (Sarter & Woods, 1995).

Proliferation of mode rich systems can result in the creation of new cognitive demands on the operator and can lead to confusion. For example, the Flight Management System (FMS) requires pilots to know which mode to use when, how to switch from one mode to another, and how each mode is set up to fly the aircraft properly (Sarter & Woods, 1995). All of these conditions make systems more difficult to handle, explain, design, and predict their behavior.

A formal specification language helps in designing complex systems satisfying a set of safety criteria, thus making the system more robust. A language like SpecTRM-RL provides tools capable of a more consistent design during the early phases of development. It is hypothesized that such languages providing features that enhance readability and can be used by the common public will help make formal methods more popular and will help engineers realize its power in systematically detecting error forms.

The following section will outline the research goals of this research. The first goal was to build a blackbox model of a complex system with a formal specification language. The second goal was to provide a more structured framework for detecting mode confusion. Chapter 1 provides a description of the model for the vertical flight control system that was used as a case study. Section 2.1 explains why the language was chosen for this case in particular. It describes the foundation behind the language and why it is suited for modeling process control systems. Section 2.2 introduces the areas of the vertical flight control system that were modeled. It also provides examples of the different elements of the model. The chapter ends with a summary of the lessons learned while building the model. Chapter 3 provides a description of the human interface criteria used as a framework for analyzing mode confusion in software design. Section 3.2 provides a review of related work in the area and describes the differences and similarities with this thesis. Section 3.3 provides a description of the human task analysis language and describes how it is formally translated into SpecTRM-RL. Section 3.4 provides a formal analysis of ambiguous interface errors. It includes recommendations over how to better design the interface in order to avoid this error type. Section 3.5 describes operator authority limits and also provides with solutions of how to apply algorithms based on the models of the system and the data structure of the state machine. Finally, general conclusions are provided in chapter 4.

1.1 Research Goals

The goal of this research is to analyze human factors problems, specifically mode confusion, that are part of the software used in most of the autoflight systems for commercial aircraft. The analysis is based on a formal methodology to systematically detect potential sources of problem. The use of SpecTRM-RL (Specification Tool and Requirement Methodology – Requirement Language) for modeling of the case studies help illustrate several points: scalability for large models, reviewability of the system specification by a general audience, and the suitability of the language to analyze mode confusion. The case study is the specification for the MD-11 vertical flight control system.

In essence, there are two goals of this thesis, one is to build the model of the entire vertical flight control system with SpecTRM-RL and the other is to provide a definition of the safety criteria for analyzing mode confusion and propose ways to systematically detect them using this model. Experience gained in this effort will also shed light in the enhancement of SpecTRM-RL for future projects.

2 Modeling the MD-11 Vertical Flight Control System

One contribution of this research is the experience gained in using SpecTRM-RL, a formal specification language, to model a large complex system. The model is complex in the sense of having many different highly dynamic components and subcomponents. To this end, the MD-11 vertical flight control system was chosen as a case study. It will be modeled by using SpecTRM-RL, which allows for different types of analysis in terms of safety. Checks for completeness, consistency, and software deviation analysis are available as part of the language.

2.1 Modeling Language

The selection of SpecTRM-RL to model a large and complex dynamic system, such as the MD-11 vertical flight control system, offered many advantages over other formal specification languages. The language is designed to support the way engineers think about control systems, and its underlying formal model is based on a finite state machine. This combination allows in depth analysis and easy development of models.

Readability is arguably the most advantageous characteristic of the language. It allows all involved parties to discuss and analyze a specification with little knowledge of formal methods. Further, experience has shown that in analyzing formal specifications for complex systems, most significant errors will be found by human experts rather than automated tools. Another added benefit was the short familiarization time needed to master the language. Although some amount of training is needed, the amount is minimal compared to other formal specification languages, such as PVS.

The types of models used in SpecTRM-RL are considered blackbox. The behavior of the system is defined solely in terms of outputs and the inputs that stimulate those outputs. By following this approach, the models do not include any information about the internal design of the system, only about its externally visible behavior.

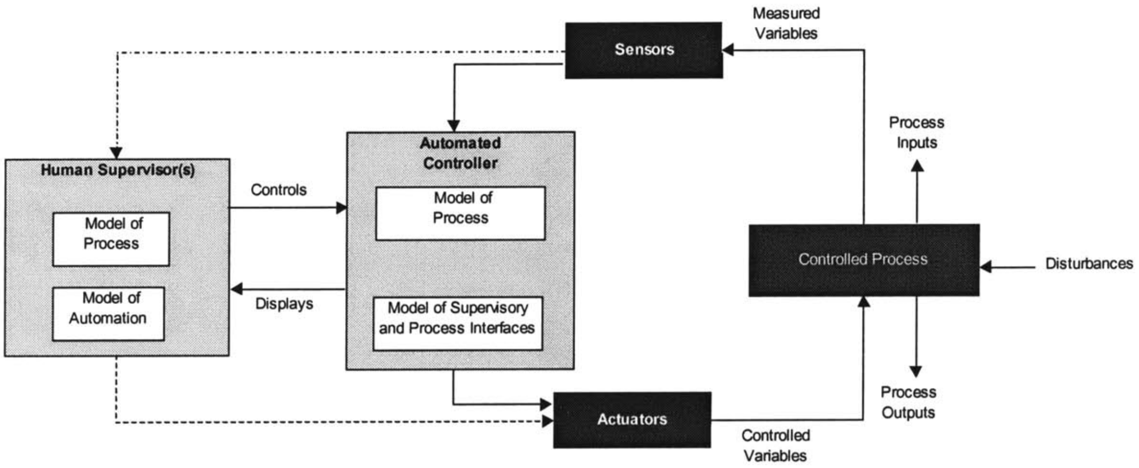


Figure 2.1. Basic Supervisory Control Loop

It is important to note that SpecTRM-RL was designed primarily for process-control systems. Therefore mapping the parts of a SpecTRM-RL model to the parts of a process control loop reduces semantic distance. A typical process-control system is illustrated in Figure 2.1. It contains four types of components: the process, sensors, displays, actuators, and controller. The blackbox model captures the controller's internal model of the process. It is argued that accidents will occur when the internal model does not accurately reflect the state of the controlled process.

The SpecTRM-RL notation is driven by the way engineers draw and define control loops. It provides a simpler graphical representation than languages such as Statecharts, PVS, and Z. Elements of the state machine formalism are hidden to the user and only those components necessary to specify input/output function are included.

Figure 2.2 shows the four main components of a SpecTRM-RL specification: (1) a specification of the supervisory modes of the controller being modeled, (2) a specification of its operating modes, (3) a model of the controlled process that includes the inferred operating modes and system state (inferred from the measured inputs), and (4) a specification of the inputs and outputs to the controller.

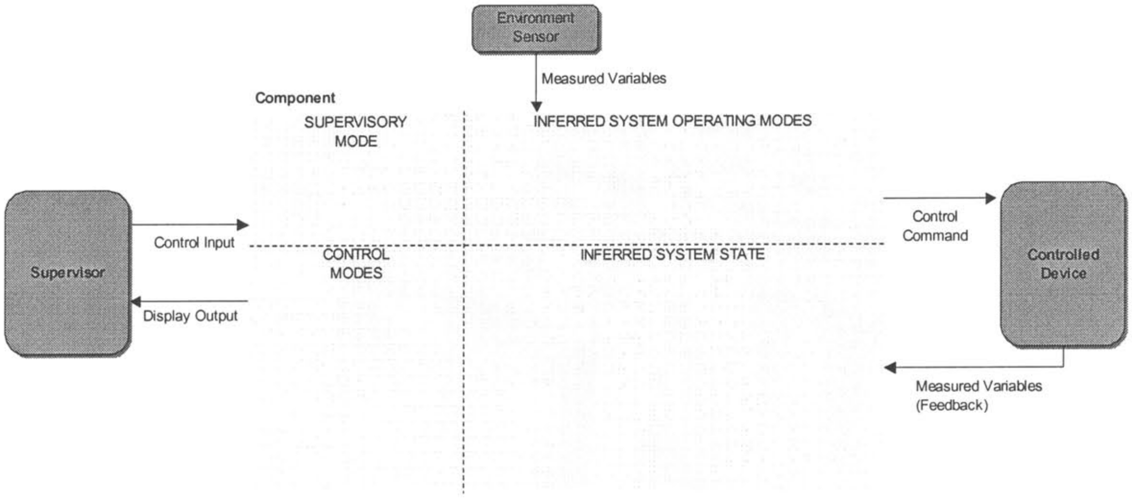


Figure 2.2. The Form of a SpecTRM-RL Specification

Every automated controller has at least two interfaces: one with the supervisor(s) that issues instructions to the automated controller (the supervisory interface) and one with each controlled system component (controlled system interface). The supervisory interface is shown to the left of the main controller model while the interface with the controlled component is to shown the right. There may be additional interfaces (shown at the top) with various environmental sensors.

The representation of the different interfaces allows analyzing the model in different ways. The supervisory interfaces provide the model of the operator controls and a model of the other ways to communicate information to the operator. The interface models are composed of the logical view that the controller has of the interfaces – which is independent of what actually occurs in the system. By having a representation of the assumed interface vs. an actual interface, the analysis can detect the effect of inconsistencies, errors, and failures. In addition, separating the physical design of the interface from the logical design will facilitate changes and allow parallel development of the software and the interface design.

Another advantage of SpecTRM-RL is the ability to specify modes. Modes are abstractions on states and are not part of the description of the blackbox behavior. They represent a logical way for users to organize the operation of the diverse set of behaviors of complex systems. To this end, modes are used as state variables that play a particular role in the state machine. SpecTRM-RL allows specifying several types of modes: supervisory modes, control modes, controlled-system operating modes, and display modes.

Supervisory modes are used in specifying information about the current supervisor of the controller and are useful when a component may have multiple supervisors at any time. For example, a flight control computer in an aircraft may get inputs from the flight management computer and directly from the pilot. Required behavior may differ depending on which supervisory mode is currently in effect. Mode-awareness errors related to confusion in coordination between multiple supervisors can be defined (and the potential for such errors theoretically identified from the models) in terms of these supervisory modes. In systems with complex displays (such as Air Traffic Control systems), it may also be useful to define various *display modes*.

The bottom left quadrant of Figure 2.1 provides information about the *control modes* for the controller itself. These are not internal states of the controller (which are not included in our specifications) but simply represent externally visible behavior about the controller's modes of operation. *Control Modes* are used in describing the required behavior of the controller. Modern avionics systems may have dozens of modes. Control modes may be used in the interpretation of the component's interfaces or to describe the component's required process-control behavior.

The right half of the controller model represents inferred information about the operating modes and states of the controlled system (the *plant* in control theory terminology). A simple plant model may include only a few relevant state variables. If the controlled process or component is complex, the model of the controlled process may be represented in terms of its operational modes and the states of its subcomponents. *Operational modes* are useful in specifying sets of related behaviors of the controlled-system (plant) model. For example, it may be helpful to define the operational state of an aircraft in terms of it being in takeoff, climb, cruise, descent, or landing mode.

2.2 MD-11 Vertical Flight Control System

The vertical flight control system of an advanced technology aircraft like the MD-11 is integrated as part of the flight management system (FMS) with the purpose of increasing safety and improving performance. It provides accurate targets and optimized controls necessary to maintain a predetermined vertical profile and provide guidance, control, and annunciation functions. It reduces costs by flying algorithmically over economized fuel paths. It also increases safety by performing critical maneuvers, annunciating alerts, and augmenting the control characteristics that make the aircraft easier to fly. This research tries to model the blackbox behavior of the automation in order to be analyzed in terms of safety.

Many tasks of an integrated flight system are distributed between the FMS and the autopilot. The FMS performs the tasks of flightplanning by compiling a four-dimensional route, defined by a lateral flightplan, a vertical flightplan, and an elapsed time trajectory. These flightplans are compiled from stored navigation databases and flight-crew entries.

Navigation is performed by the FMS by identifying aircraft position relative to fixed points on the surface of the earth. A lateral position, vertical position, and elapsed time are computed by a combination of data from ground-based transponding radios, radar, and aircraft motion sensors.

Guidance is performed by the FMS by determining the appropriate altitude, speed, thrust, and heading required to maintain the current leg of the flightplan. These targets are determined by a comparison of aircraft position (Navigation) to the desired profile (Flightplanning) and may take into account temporary deviations from the flightplan due to weather, traffic, equipment failures or on-board emergencies.

The autopilot controls the aircraft by synchronously adjusting the pitch, roll, and yaw control surfaces and the throttle position, to instantaneously maintain the desired aircraft trajectory defined by FMS Guidance.

Flight management of the vertical profile is modeled by a set of vertical guidance operational procedures. This set of procedures represent all the necessary maneuvers required by Air Traffic Control (ATC) and the flight-crew to manage the aircraft trajectory in the vertical profile. Each Vertical Guidance Operational Procedure is defined by a set of attributes that distinguishes them. They are described below.

The objectives/strategies of a vertical guidance operational procedure represent the goals, manner of operation, or the specific maneuver that can be achieved by this procedure. This attribute distinguishes one operational procedure from another.

The engagement criteria identifies aircraft position in the vertical flightplan of the state of the aircraft when the operational procedure is invoked. It accounts for pilot instructions received from the flight control panels, equipment status, and the position of the aircraft relative to the desired vertical flightplan.

The altitude target identifies the level altitude that shall be maintained or acquired by the aircraft to satisfy the objectives of the procedure. The altitude target, determined by the selection of one of the altitude scenarios, may be the pilot selected clearance altitude or an altitude constraint specified in the vertical flightplan.

The control mode identifies the parameters that shall be acquired and maintained by the integrated pitch/thrust control mode. The attributes of the control modes include the methods for computation of the elevator and throttle commands (typically closed-loop feedback with a predictive term) and the control error criterion specified by the advisory circular of the FAA. The control mode is determined by the selection of one of the pitch/thrust combinations available in the autopilot.

The speed target identifies the airspeed and mach targets that shall be acquired and maintained to satisfy the objectives of the procedure. The airspeed and mach targets, determined by the selection of one of the speed scenarios, may be pilot selected airspeed or mach from the flight control panel, an optimum speed defined by the FMS, or a speed specified in the vertical flightplan.

The vertical speed target identifies the rate of descent that shall be acquired and maintained in order to satisfy the objectives of the procedure. The vertical speed targets, determined by the selection of one of the vertical speed scenarios, and may be the pilot selected vertical speed from the FCP, or a rate of descent identified for a specific procedure.

The model that was built of the automation is based on a real-time selection of a vertical guidance operational procedure and the subsequent selection of the appropriate target and control-mode scenarios. Figure 2.3 shows this selection logic. The applicable procedure is selected responding to the logic of the control laws and the conditions of the engagement criteria scenarios. The decisions consider the current states of the vertical flightplan (including flightcrew instructions) and feedback of the navigation aircraft position. Also based on the selection of the procedure, the vertical flightplan and aircraft position are chosen as well as the altitude target, airspeed/mach target, vertical speed target and control-mode.

For example, for the descent path operational procedure the altitude target status defines the control logic for the aircraft altitude and the flight control computers modes. The vertical guidance type and descent/approach path status are also included as part of the definition and logic of the system. They define the capture hold criteria and the engagement of the vertical profile.

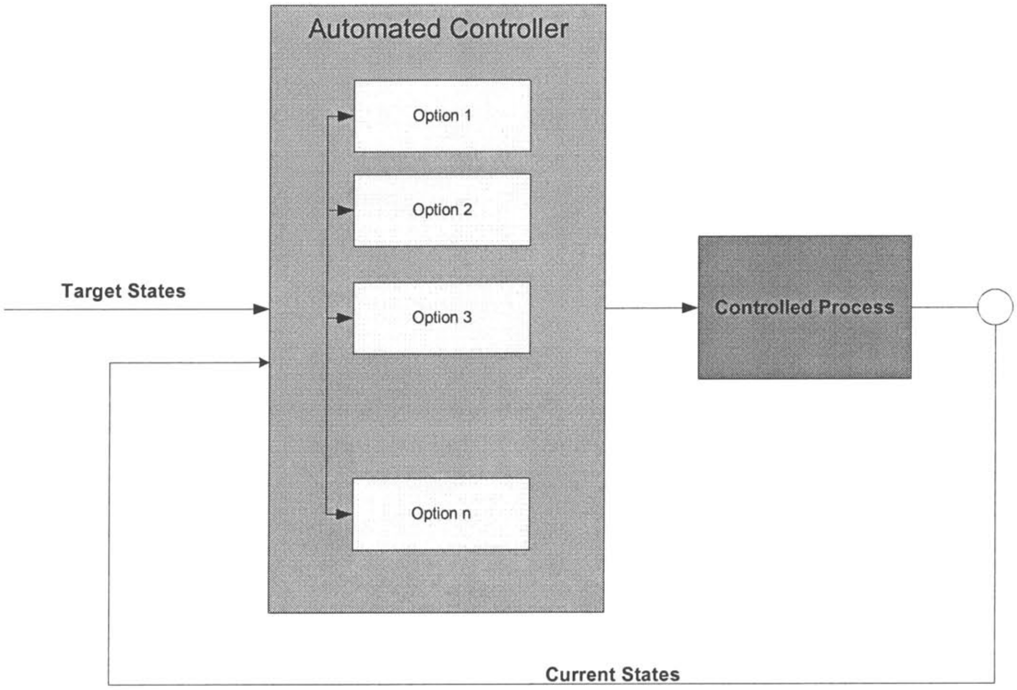


Figure 2.3. Closed Loop Feedback

Figure 2.3 shows visualization of how the model of the automated controller works. It selects the appropriate vertical guidance operating procedure evaluating the current situation. Each of the options has an associated set of requirements for the selection of the option.

There are three main areas covered in our model of the vertical flight control system: interpretation, guidance, and annunciation. Figure 2.4 shows an overall view of the model. The interpretation process generates and correlates all the necessary data required by the guidance, control, and annunciation processes. Using the aircraft position relative to its vertical profile and any operational commands from the pilot, the *guidance* function determines appropriate altitude, speed, thrust, path, and pitch targets as well as the integrated pitch/thrust control mode necessary to maintain the desired trajectory. Finally, *annunciation* provides information to the FCC (flight control computer) to be displayed in the cockpit, based on the guidance and control functions. This information includes the flight mode, speed target, and altitude targets of the aircraft. Figure 2.4 shows the overall view of the different areas covered by our model.

Figure 2.5 shows the SpecTRM-RL model of the system. Because the vertical flight control system is too large to show on one page (or one screen), the specification is hierarchically decomposed and only state variables are shown. Figure 2.5 also shows the possible values that the *Flight Phase* state variable can assume. In reality, every state variable (including operating modes) listed on the system model is associated with a similar set of values. However, only the list of names for state variables is shown due to space limitations.

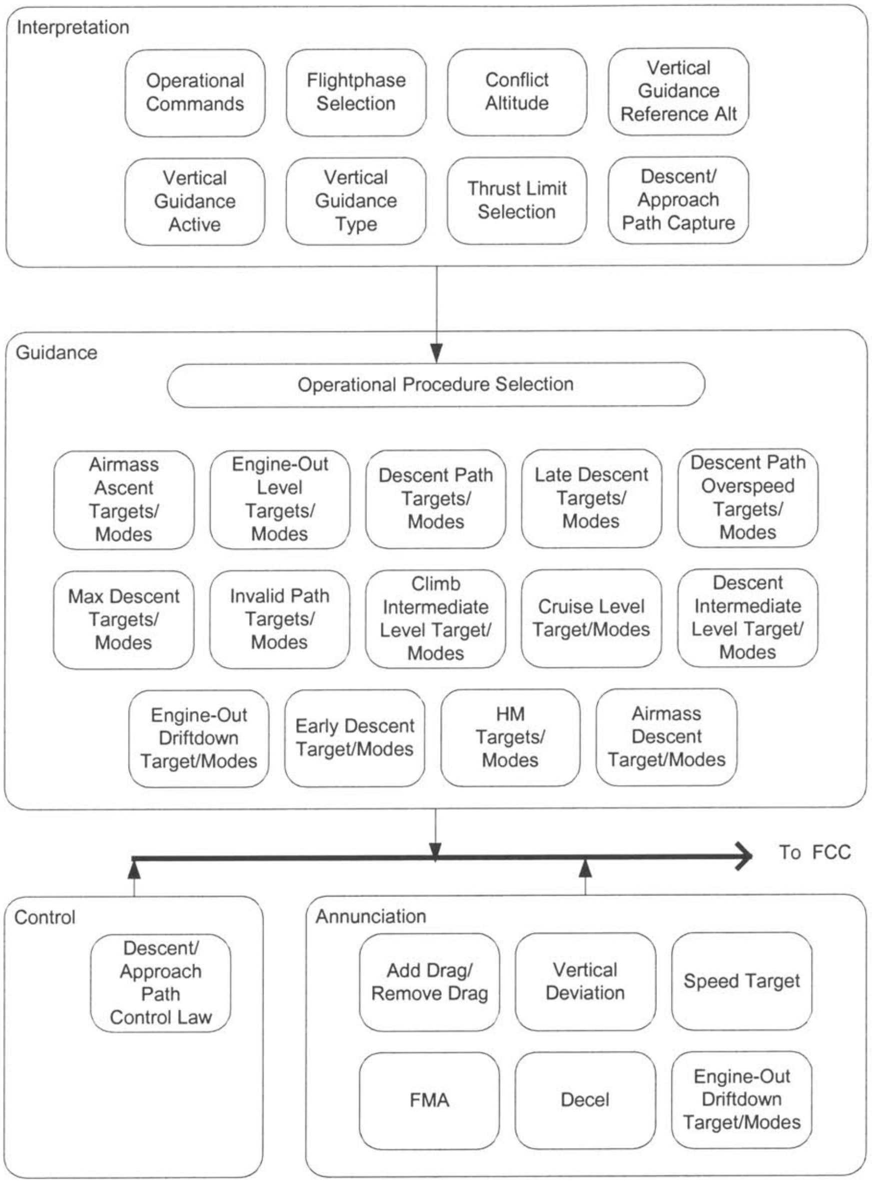

Interpretation
Operational Commands
Flightphase Selection
Conflict Altitude
Vertical Guidance Reference Alt
Vertical Guidance Active
Vertical Guidance Type
Thrust Limit Selection
Descent/ Approach Path Capture
Guidance
Operational Procedure Selection
Airmass Ascent Targets/ Modes
Engine-Out Level Targets/ Modes
Descent Path Targets/ Modes
Late Descent Targets/ Modes
Descent Path Overspeed Targets/ Modes
Max Descent Targets/ Modes
Invalid Path Targets/ Modes
Climb Intermediate Level Target/ Modes
Cruise Level Target/Modes
Descent Intermediate Level Target/ Modes
Engine-Out Driftdown Target/Modes
Early Descent Target/Modes
HM Targets/ Modes
Airmass Descent Target/Modes
To FCC
Control
Descent/ Approach Path Control Law
Annunciation
Add Drag/ Remove Drag
Vertical Deviation
Speed Target
FMA
Decel
Engine-Out Driftdown Target/Modes

Figure 2.4. Overall view of the Vertical Flight Control System.

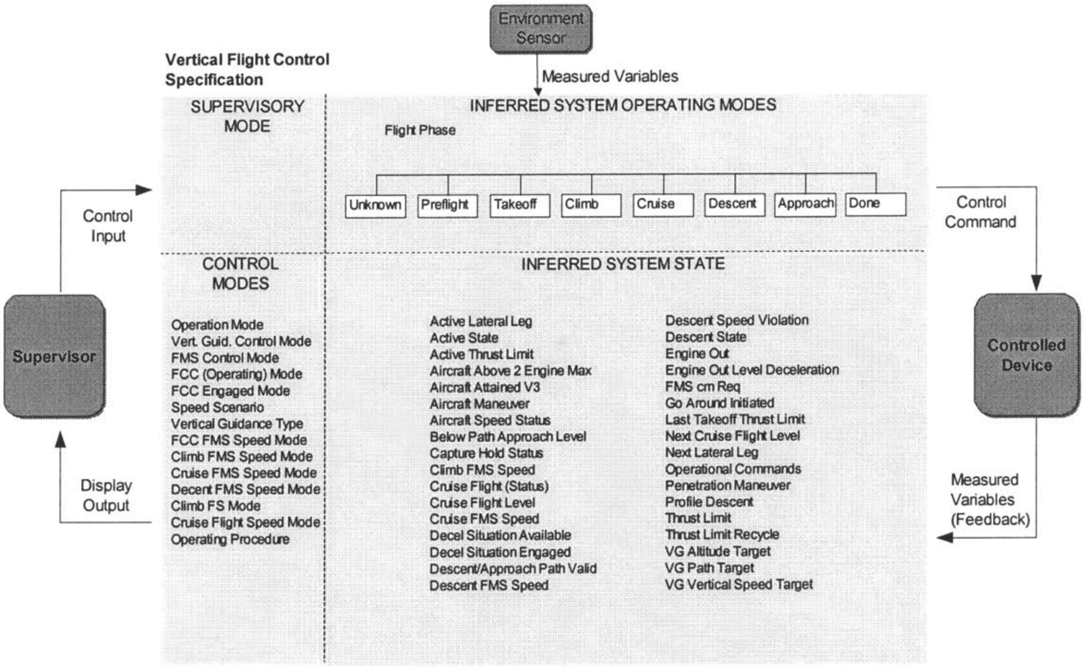


Figure 2.5. MD-11 SpecTRM-RL Model

The SpecTRM-RL model of the MD-11 vertical flight control system is composed of the graphical notation along with detailed specifications for output messages or commands, modes and inferred states variables, input and output variables, macros, and functions. The graphical notation has been described and each of the other features is now described.

2.2.1 Output Message Specification

Figure 2.6 shows an example for the specification of an output. The general format for the output specification includes information regarding: destination of the output; acceptable values; timing behavior, including any initiation delay or completion deadline along with any required exception-handling behavior if the deadlines cannot be met, output load and capacity limitations, etc.; feedback information about how the controller will determine that the output command has been successfully implemented; and the identity of any other output commands that reverse this output (Leveson , 2000).

The output variable for the *Target Altitude* determines the current leg of the aircraft's path. The conditions under which outputs are assigned values are described using a tabular representation called AND/OR tables. Each AND/OR table is divided into two parts, *Control Modes* and *State Values*. The control modes section describes the value of the control modes necessary for the transition, while the state values section describes the values of and conditions on inputs and state variables. This distinction allows the reader to better understand each mode of the system's behavior, and it was found helpful in detecting specification errors – particularly omissions.

AND/OR tables are concise representations of propositional logic (in disjunctive normal form) that are easily read and interpreted. The far left column of the table lists logical (boolean) phrases. Each of the other columns is a conjunction of those phrases and contains the logical values of the expressions (a '*' denotes "don't care"). A column evaluates to *true* if all of its elements are true. If one of the columns is *true*, then the table evaluates to *true*. For example, the *Target Altitude* output Figure 2.6 would be *Vertical Guidance Climb Target Altitude* if the *Operating Procedure* is *Airmass Ascent* and the current flight phase is *Takeoff* or *Climb*, OR if the *Operating Procedure* is *Climb InterLev* and the current flight phase is either *Takeoff* or *Climb*. This type of representation allows summarizing extremely complex and long requirements.

Output Variable

Target Altitude

Type: INTEGER
Destination: TBD
Initiation Delay: 0 milliseconds
Completion Deadline: TBD
Exception Handling: None specified
References: N/A

Feedback Information:
Variables: UNDEFINED
Values: UNDEFINED
Min time between outputs: 10 MHz
Max time between outputs: UNDEFINED
Exception Handling: None Specified

:= Vertical Guidance Climb Target Altitude **IF**

TRIGGERING CONDITION

Control Modes	Vertical Guidance Operating Procedure IN_STATE Airmass Ascent	T	T	*	*
	Vertical Guidance Operating Procedure IN_STATE Climb InterLev	*	*	T	T
State Values	Flight Phase IN_STATE Takeoff	T	*	T	*
	Flight Phase IN_STATE Climb	*	T	*	T

:= Active Cruise Flight Level **IF**

TRIGGERING CONDITION

Control Modes	Vertical Guidance Operating Procedure IN_STATE Airmass Ascent	*	T	*
	Vertical Guidance Operating Procedure IN_STATE Climb InterLev	*	*	T
State Values	Flight Phase IN_STATE Cruise	*	T	T
	Active Cruise FL Valid ()	T	*	*

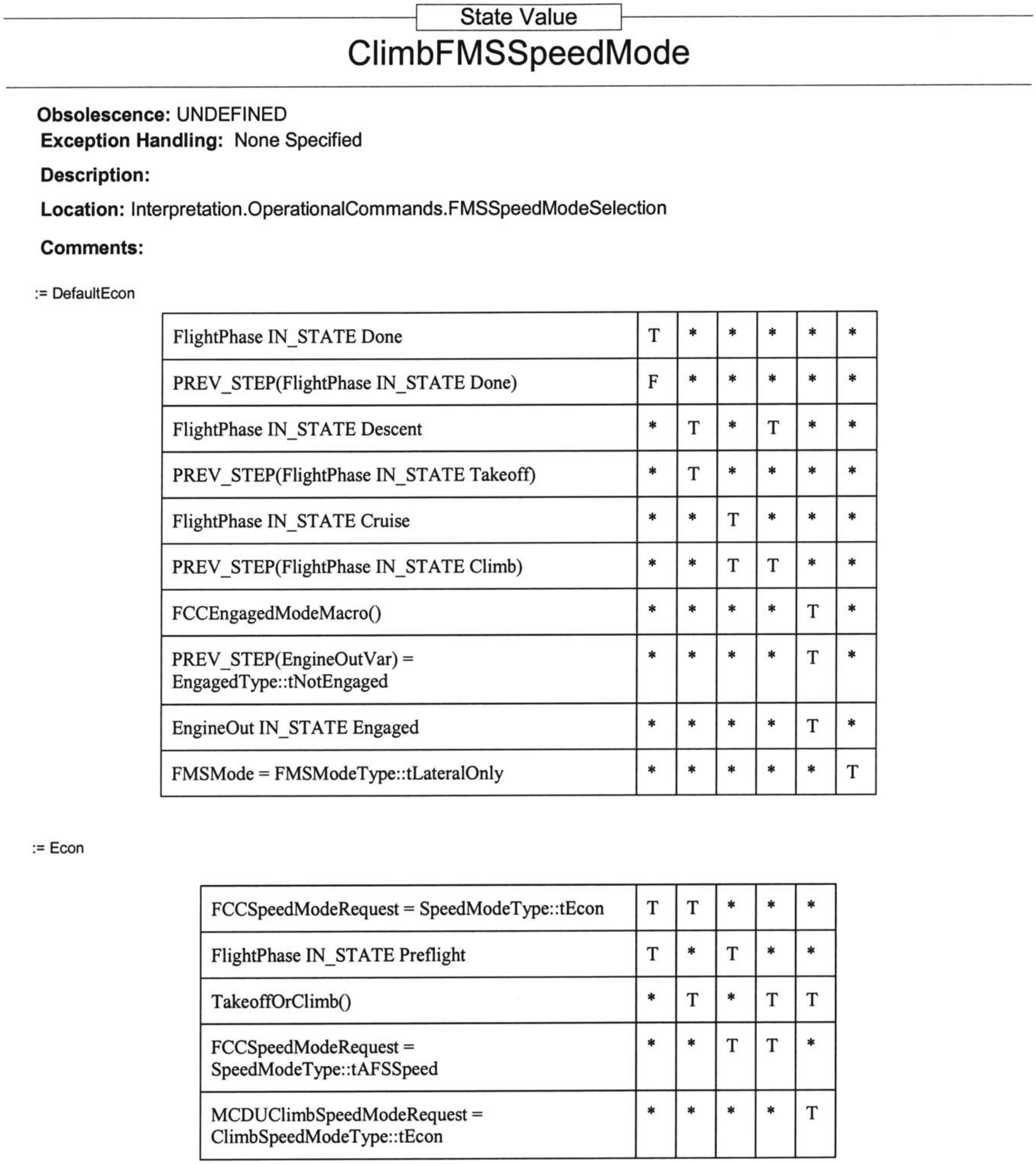
:= Vertical Guidance Descent Altitude **IF**

TRIGGERING CONDITION

Control Modes	Vertical Guidance Operating Procedure IN_STATE Cruise	*	T
State Values	Step Climb IN_STATE False	*	T
	Clearance Altitude < Active Cruise FL – 250	*	T
	Vertical Guidance Descet Target Alt != -1000	T	T
	Active Operational Procedure Valid()	T	*

Notes:
The Vertical Guidance Descent Target Altitude is assigned a default of –1000 ft when no other Descent Alt Constraints are entered in the flight plan.

Figure 2.6. Example Output Variable



State Value

ClimbFMSSpeedMode

Obsolescence: UNDEFINED
Exception Handling: None Specified

Description:

Location: Interpretation.OperationalCommands.FMSSpeedModeSelection

Comments:

:= DefaultEcon

FlightPhase IN_STATE Done	T	*	*	*	*	*
PREV_STEP(FlightPhase IN_STATE Done)	F	*	*	*	*	*
FlightPhase IN_STATE Descent	*	T	*	T	*	*
PREV_STEP(FlightPhase IN_STATE Takeoff)	*	T	*	*	*	*
FlightPhase IN_STATE Cruise	*	*	T	*	*	*
PREV_STEP(FlightPhase IN_STATE Climb)	*	*	T	T	*	*
FCCEngagedModeMacro()	*	*	*	*	T	*
PREV_STEP(EngineOutVar) = EngagedType::tNotEngaged	*	*	*	*	T	*
EngineOut IN_STATE Engaged	*	*	*	*	T	*
FMSMode = FMSModeType::tLateralOnly	*	*	*	*	*	T

:= Econ

FCCSpeedModeRequest = SpeedModeType::tEcon	T	T	*	*	*
FlightPhase IN_STATE Preflight	T	*	T	*	*
TakeoffOrClimb()	*	T	*	T	T
FCCSpeedModeRequest = SpeedModeType::tAFSSpeed	*	*	T	T	*
MCDUClimbSpeedModeRequest = ClimbSpeedModeType::tEcon	*	*	*	*	T

Figure 2.7. State Variable Example

2.2.2 State Variable Definition

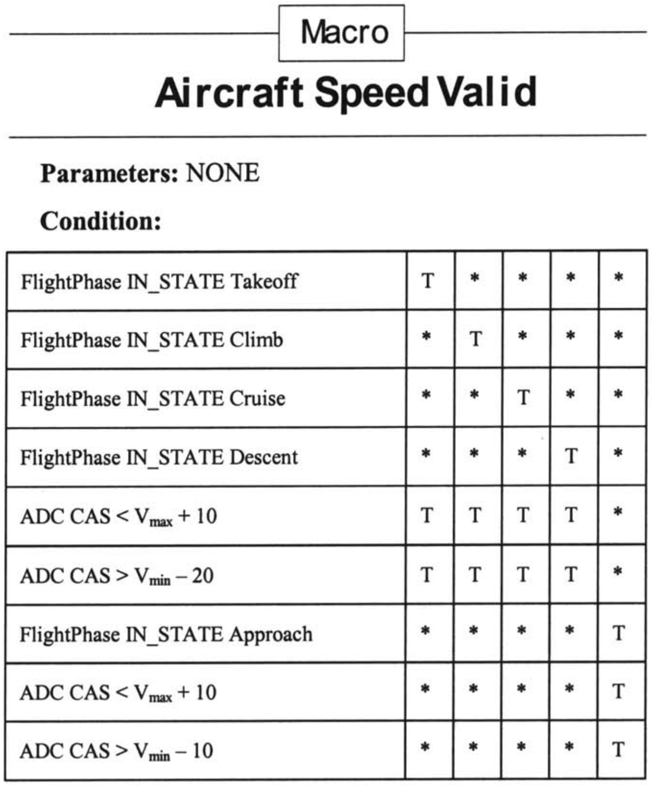
State variables were used in the model to represent different modes of operation of the system. Their values were inferred from the values of input variables or from other state variable values. Figure 2.7 shows only two of the possible values for the state variable *ClimbFMSSpeedMode.* The possible values shown are *Econ* and *DefaultEcon*. During the climb portion of the profile, the Auto-Flight System (AFS) controls thrust from FMC thrust limits and speed targets via the Flight Control Computers (FCCs). The aircraft climbs at climb limit thrust to each altitude constraint, flies level until past the constraining waypoint using an appropriate speed target, and then resumes the climb at climb limit thrust. The model represents the logic for those situations.

Not shown is the description of the *unknown* value for this state variable. In SpectRM-RL all definitions of state variables must include an unknown state. This corresponds to the default value upon startup or upon specific mode transitions. The purpose of this unknown state value is to force resynchronization of the model with the outside world after an interruption in processing inputs.

Most of the state variables were used to represent the operational procedures discussed before. The use of AND/OR tables captured the logic and they included the operational behavior of the aircraft under the jurisdiction of the FMS Speed, navigation, and vertical profile modes. Those operational procedures define the pitch/thrust axis control strategy, altitude, path, speed, and vertical speed targets required to achieve the desired objectives.

2.2.3 Macros and Functions

The use of macros in SpecTRM-RL allows the organization of the model in logical pieces for easier reference: it simplifies the complexity inherent in large models where conditions are re-used. By definition, macros are simply named pieces of AND/OR tables that can be referenced from within another table. For example, Figure 2.8 shows a simple macro that was created for this model.



Macro

Aircraft Speed Valid

Parameters: NONE

Condition:

FlightPhase IN_STATE Takeoff	T	*	*	*	*
FlightPhase IN_STATE Climb	*	T	*	*	*
FlightPhase IN_STATE Cruise	*	*	T	*	*
FlightPhase IN_STATE Descent	*	*	*	T	*
ADC CAS < V_{max} + 10	T	T	T	T	*
ADC CAS > V_{min} – 20	T	T	T	T	*
FlightPhase IN_STATE Approach	*	*	*	*	T
ADC CAS < V_{max} + 10	*	*	*	*	T
ADC CAS > V_{min} – 10	*	*	*	*	T

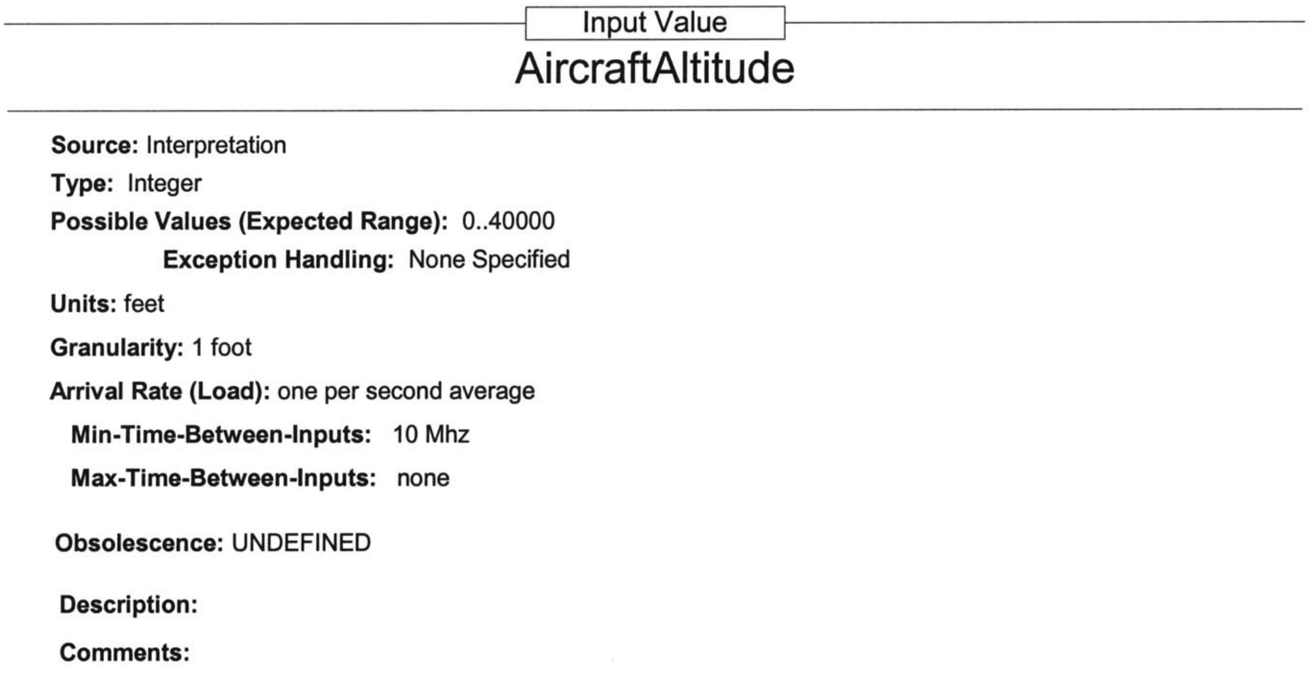
Figure 2.8. Example Macro

The *Aircraft Speed Valid* macro will evaluate to *true* if the corresponding AND/OR table evaluates to *true*. Macros are very convenient for expressing hierarchical abstraction and enhance hierarchical review and understanding of the specification. In order to handle very complex and large models, macros are almost a required feature for humans.

Functions are similar to macros in the sense that they are specified in the same way as macros with the only difference that they return a numeric value as an answer.

2.2.4 Input Variable Definition

There are several features in SpecTRM-RL that provide information regarding inputs. The user can define arrival rates, exceptional-condition handling, data-age requirements, etc. Figure 2.9 shows an example of an input definition for the aircraft altitude.



Input Value

AircraftAltitude

Source: Interpretation

Type: Integer

Possible Values (Expected Range): 0..40000

Exception Handling: None Specified

Units: feet

Granularity: 1 foot

Arrival Rate (Load): one per second average

Min-Time-Between-Inputs: 10 Mhz

Max-Time-Between-Inputs: none

Obsolescence: UNDEFINED

Description:

Comments:

Figure 2.9. Example Input Variable

Inputs in our model were organized according to their functionality. For example, the following were found to be inputs related to the Multiple Control Display Unit (MCDU) and they where grouped together: Offside MCDU InitKey Pressed, Onside MCDU InitKey Pressed, MCDU ButtonPush, Onside MCDU Discrete, Offside MCDU Discrete, TL From MCDU TLPage, Final Approach Prompt, Alt TO TL Selected on MCDU TLPage, MCDU Climb Speed Mode Request, MCDU Cruise Speed Mode Request, MCDU Descent Speed Mode Request, and Assumed Temp Entry on MCDU TLPage. This exemplifies the need to make this organization an easier task. The number of input variables in a complex model can grow and can become very difficult to handle not only during the design of the system but also during the testing period. During the simulation the number of input variables were so large that one of the only ways to observe them was to simplify the model by dividing them into sub-modules. After division into modules, observing how changes in inputs were affecting the model was manageable.

2.3 Lessons Learned

One of the first encouraging findings is the time it took to build the model. It is estimated that it took approximately eight person months to create. Considering the lack of experience or knowledge about either SpecTRM-RL or the MD-11 flight management system and the lack of sophisticated tools to assist in the development, this is a good result. It is believed that the time could be significantly reduced given more appropriate tools.

Many initiatives helped in dealing with complexity of the model from a user's cognitive point of view. Despite the improvements of the language in relation to previous versions, cognitive manageability remained a concern throughout the building of the model. Since experience in building complex systems with formal specification languages is scarce, the author had to experiment with new techniques.

The use of macros was regarded as an effective technique in dealing with complexity of the system. It served the purpose of organizing the information into manageable chunks in order to help build a hierarchical abstraction of the system. By having a way of grouping similar conditions or conditions used by similar procedures, the language allows the user to simplify his other mental model of the system and model the system with more confidence. In addition, because the model was build by a team of students, the macros were shared during the distributed development. It allowed people to work in different areas of the model but use the same macros. In case a change was needed, then only the macro needed to be changed. Keeping control of the differences in similar macros is also an essential future feature of the language. Sometimes macros will differ in a line or two, or in the order of the condition definitions, making for confusion while reading them.

One of the tools that would have reduced the time it took to build the model is the availability of a data dictionary. This tool could have been helpful for organizing the state variables, inputs, outputs, and macros in regard to defining their names and location or just for simply organization purposes. It could have provided better ways to control the names used thereby reducing confusion – the user would have been able to determine if the name was used before and where or if it was misspelled. SpecTRM-RL provided error messages if states were not defined properly or improperly referenced during a declaration but only after the model had been build and not during the construction of the model. By incorporating this data dictionary as part of the actual set of tools, designers can save time, effort, and achieve a more consistent design.

Cognitive manageability was also a concern when modifying the specification. It was often very difficult to consider and evaluate the ramifications of a potential change. In addition, the simulation tool itself was limited in this regard. It did not allow much experimentation on how to visualize changes in the dependencies among state variables.

Verifying the correctness of the model was also a painstaking experience. The model was easy to read and (when looked at in reasonable chunks) understand, but maintaining an accurate mental model of the entire system proved to be a challenging, if not impossible, task. It is hypothesized that the use of collapsible columns and rows in tables would have helped in this area. By allowing the user to select a specific area of the model that she would have liked to concentrate her attention on, discoveries of errors and revisions of dependencies of variables would have been easier to conduct. To accomplish this, a *Location* element could be included as part of the definition of every state variable. The location element would define the particular area and type of supervisory mode the state variable is being used in the model. This would allow highlighting a specific area of relevance and reducing the number of columns and rows to just the ones of particular interest, making the model more manageable.

There are still tools needed to assist humans in dealing with the complexity of any complete specification. However, some simple tools available were found very useful. For example, the high-level, graphical view of the specification helped keep the overall structure in mind, and the fact that SpecTRM-RL is executable was useful in error-checking the evolving specification. A consistency and completeness tool identified input conditions that were either not accounted for in our specification or led to an ambiguous system response (i.e., nondeterministic behavior). In summary, SpecTRM-RL proved to be a valuable tool in the development of the system specification.

3 Human Machine Interface Design Criteria

3.1 Introduction

The motivation behind building a model of the vertical flight control system was the ability to analyze the model for safety – in particular mode confusion errors. The goal is to provide a methodology to design the automation so that it will support human actions. It was intended that by having a model of the automation[1], design flaws could be detected. Many of these errors are caused by designers focusing on the software mapping of inputs to outputs, on mathematical models of required functionality, and on the technical details and problems internal to the computer: Little attention is given to evaluating software in terms of whether it provides transparent and consistent behavior that supports operators in their monitoring and control tasks. The goal was to use the SpecTRM-RL language and the set of tools to create a model of the automation and evaluate a proposed methodology for integrated design of complex systems. It is during these early stages of design that changes can be made relatively easily.

In this effort, one of the first steps was to determine an initial safety criterion that would be able to detect or prevent automation features that can lead to operator mode confusion. Leveson has identified six types of potential design features: inconsistent system behavior, indirect mode transitions, lack of appropriate feedback, operator

[1] Although a version of the MD-11 FMS was used to derive the example for this case study, nothing in this thesis should be taken as applying to that aircraft's actual automation as the specification used is not that of that from the current MD-11 software.

authority limits, unintended side effects, and ambiguous interfaces (Leveson, 1997). This chapter will suggest algorithms that will explore ways to detect them.

3.2 Analysis Approach

The hierarchical representation of modes parallels the hierarchy used in modeling control systems. This is no coincidence. This representation of modes will be used to detect error forms. In this sense, errors can be analyzed as breakdowns in the model of the controlled system. Whenever the model is updated and consistent with the actual system state, no conflicts arise. At the time of a divergence, erroneous control commands result that could lead to accidents. This thesis will look for ways to make this comparison.

This thesis is proposing a methodology for analysis based on many areas that are considered critical. This methodology includes a model of the user actions and a corresponding model of the automation.

Leveson and Palmer studied ways to identify design flaws that are commonly reported by the ASRS. One result of this case study was a recognition that such mode confusion errors could only be identified if the software (automation) model was augmented by a simple model of the controller's view of the software's behavior (a user model) – the formal software specification was not enough.

The closest approach to this that of Degani and of Harrison and Fields. Degani provides a different classification of modes and of mode confusion detection criteria. His classification of mode confusion detection criteria is a subset of Leveson's. Degani also developed a task modeling framework, known as OFAN, which is based on the Statecharts language. Previous experience in using Statecharts on real systems found it not to be scalable for large and complex systems (Leveson, 1994). Harrison and Fields (1995) use CSP models to attempt to achieve similar goals, but this language is very formal and probably not usable without extensive training in discrete mathematics.

Javaux takes a more psychological approach. He uses a finite state machine to describe a cognitive mental model, which he uses to identify potential instances of mode confusion. In contrast, this thesis does not try to model human cognition or human mental models. Instead, it uses a representation of the blackbox behavior of the automation that the user expects and depends upon and the required steps needed to complete a given task. Modeling the actions involved in an operator task potentially allows analyzing the interaction of the operator with a formal model of the rest of the system.

Vakil and Hansman have proposed the use of predictability as a measure of system complexity, which they claim is directly related to mode confusion. Their work also suggests approaches to mitigating complexity-driven issues in commercial flight systems.

Other researchers have applied Leveson's safety criteria for mode confusion and tried to use formal methods techniques (e.g., model checking) to detect these error forms in system specifications.

The study of errors in this thesis is based on the premise that by having a formal model of the automation – as described before – the systematic detection of faults is made easier. Because errors result from inconsistent models of the controlled process, the representation of those models becomes a critical issue. Models may diverge either because they were incorrect or incomplete to begin with (they do not adequately reflect the behavior of the controlled system) or because they are improperly updated due to incorrect feedback about the state of the modeled system. Note that there are several sources of inconsistency due to improper feedback.

The following sections will intend to explicitly specify and validate these models. It is believed this will allow engineers to identify and eliminate error-prone features of automation and will be able to assist in task analysis and development of operator training and reference materials.

3.3 Human Task Analysis

In order to complement the analysis of mode confusion, a task analysis needs to be conducted. To solve this problem, a model of the human actions needs to be created. Previous work in this area had identified a visual formalism for task analysis that was easily translated into SpecTRM-RL. The work of Brown and Leveson (1998) concentrated in modeling operator tasks in the area of air traffic control. The case chosen in this thesis is different in the sense that the interaction with the computer needed to be included, in contrast to just building a model based on the interaction among operators. Therefore the language was modified in order to accommodate for the new needs (Rodriguez et al., 2000).

The use of a visual formalism helps to organize the safety analysis. A task analysis can be added to the model of the actual system so that it can be referenced and analyzed easily. It provides a model of the different areas that were found critical – like display information – for analyzing human-computer interaction. Three models were created:

1) A *user model* of the automation behavior: this model is necessarily a simplification or abstraction of the actual possible automation behavior (which is often not completely understood even by the designers),

2) An *operator task model* resulting from a task analysis that identifies the major tasks of the human controller and then breaks these down into subtasks, eventually specifying the tasks down to the level of the key presses, voice communications, display cues, etc. involved in performing the task, and

3) A detailed specification of the *blackbox automation behavior*, that is, the blackbox behavioral requirements of the automated system.

A case study was fully analyzed by Rodriguez et al. and will not be covered here. The contribution to this thesis is to formally describe this translation from the visual formalism to SpecTRM-RL and describe how it can be used in the analysis of mode confusion.

The components of the modeling language are shown in Figure 3.1. Steps required to complete a task are represented by *states*. A *transition* is defined as the process of changing from one state to the next. Conditions that trigger transitions are called *events* and *action* describes a result or output from the transition. A *communication point* links different models together. In this case, the human task model communicates its outputs (or actions) via a communication point to the system model.

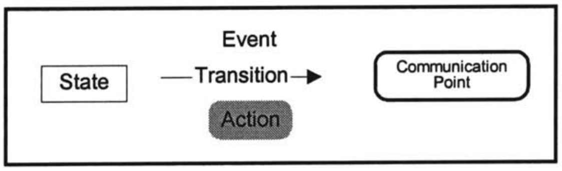


Figure 3.1. Visual Task Modeling Language Components

A *state* is represented by a square box. A *transition* is represented by an arrow from one state to the next, and *events* are shown in text above the transition; *actions* are represented by text with gray shade beneath the transition. Finally, c*ommunication points* are round boxes and refer to another component in the model (Brown and Leveson, 1998).

Figure 3.2 shows the user and pilot task models that were created. Relevant parts of a display model are also included that show the feedback regarding the state of the automation provided to the pilot. The figure shows parts of the tasks that the pilot needs to take into consideration as described by the Pilot's Guide. In particular, it is referring to the early descent operating procedure.

Benefits of translating the visual formalism into a formal language like SpectRM-RL allows further analysis in terms of safety. The following paragraphs will describe those elements of the visual formalism and how they are translated.

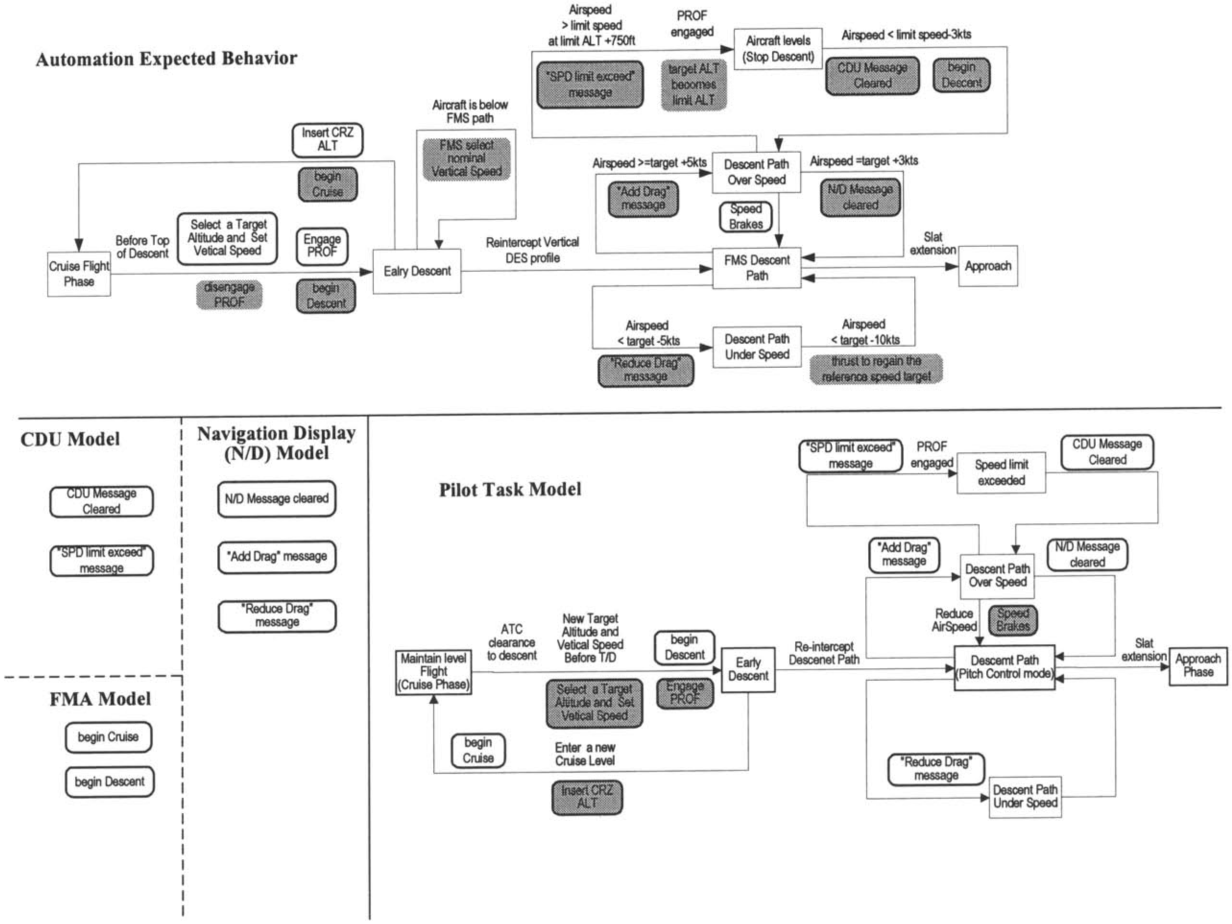


Figure 3.2. User and Task Models

Each component of the task model is represented as a component in SpecTRM-RL, consisting of a single state variable. This state variable can assume one of a set of finite values at any time, namely the states of the task model component. The default value of the state variable will be the first state of the task model component.

Conditions and Events can be translated as Inputs in the SpecTRM-RL model. All conditions and events in the task model will be represented as input variables. By default, these input variables could be defined as boolean (e.g. ATC_Clears_Descent = {True, False}), though they may be other types when other types will be appropriate (e.g. enumerated, integer). The declaration of such variable types could be made directly into the task analysis model in order to facilitate the construction of the SpecTRM-RL model or taken from the existing system model. Events could be organized by a data dictionary feature that will allow the display of current values and will ease the adding/deleting of possible cases for the events and conditions.

The representation of the actions becomes a more complicated issue. Actions are represented as input variables in SpecTRM-RL models, but may additionally be represented as output variables to other models, if they are part of a communication point. This depends on the location of the action with regards to the model in which is currently operating. The input variable needs to notify the component that the action (e.g. Pilot_Engages_PROF) has been performed, but the output variable may also be necessary if this action is communicated to other components. The safety criteria elements for timing could also be included directly into the task analysis modeling language allowing the designer to insert timing requirements and max/min conditions among others. The definition of an action could be very helpful in analyzing conditions like "forcing functions" defined formally as conditions where the inputs from the operator are ignored. By having these models, the analyst can visualize those conditions and see what are the implications.

Transition definitions between components are represented by AND/OR tables. They denote the triggering conditions. As part of the definition, critical assumptions – those where the improper performance of the software can have severe consequences – can be included (Leveson, 1995). This will assure trigger event completeness and help build a more robust system.

Display signals are represented as output variables in the SpecTRM-RL component and as well are boolean by default (e.g. “ADD DRAG” displayed = {True, False}). Display outputs are triggered when a certain transition is taken in the task model. Hence, a display variable is activated when the state variable representing the component makes a certain transition from one value to another.

Communication points denote a link between an output variable of one component and an input variable of another. Each input/output variable associated with the same communication point is defined with the same name. If an action is part of a communication point, it must be represented not only as an input variable for the component, but as an output variable as well. The names of such input/output variables must be distinct, as SpecTRM-RL does not allow two variables to share the same name. Communication points are also valuable elements to be analyzed for safety criteria issues related to human-computer interaction. Further analysis of safety criteria issues could be added to the formal definition of a communication point in order to incorporate safety into the construction of these models.

The following sections will describe selected human-machine interface criteria applicable to the detection of mode confusion. There are based on the work by Leveson that identified design features of blackbox automation behavior that can lead to operator mode confusion or mode awareness errors. Examples from the model are provided as well as explanations on how the design criteria was applied using SpecTRM-RL.

3.4 Ambiguous Interface Errors

This classic mode confusion error form occurs when the computer interprets user-entered values different than intended or it maps multiple conditions onto the same output depending on the active control mode (Leveson, 1997). As a result, the operator interprets the interface erroneously.

An example of this type of error common to many computer occurs in some word processors where the user may think they are *in insert mode* but instead are in *command mode* and their input is interpreted differently than they intended. There were many instances in the development of the model where this potential mode confusion was encountered and selected examples will illustrate how they were modeled.

The following section describes how some of the potential error prone features were discovered in a model of the MD-11 flight control system and how to create algorithms to detect those error prone features.

3.4.1 Design Criteria for Detecting Ambiguous Interfaces

There are several design criteria described by Leveson to detect this classic form of mode confusion error. These design principles are intended to be incorporated into the SpecTRM-RL model in order to be systematically detected and conveyed to system designers during early stages of development. One of the advantages of using SpecTRM-RL is its mode classification feature, which makes it easier to incorporate the safety design criteria in simple ways that are visually detectable and manageable.

One of the first design constraints discussed by Leveson is that any mode used to control interpretation of the supervisory interface should be annunciated to the operator – that is, it should be part of the displays interface in the modeling language. In describing such a situation in the model of the automation, the current operating mode of the automation should be annunciated and be part of the display interface, as well as being part of the operating modes. Many state variables were found not to satisfy this design criteria.

An example has been chosen to illustrate this point. Detection of the failure to comply with the design criteria is assisted by the use of a tool that provides a visualization of the relationships among the different state variables in the model. The example was chosen after conducting the human task analysis and it is based on the Pilot's Guide and the analysis of indirect mode transitions discussed by Rodriguez et al., 2000. Without this analysis it would have been impossible to detect the relationships and the potential sources of problems with the interface.

The document for the vertical flight control system – which was the source of information for building the model – was written in plain English and did not provided any linking among the different components. Therefore while building the model a distinction needed to be made between the control modes and the display modes – something that was not originally included as part of the English specification. The translation to the SpecTRM-RL model became a challenge since most of the state variables referred to operating modes and it was impossible to foresee how they were used by the other display components – namely FMA, Navigation Display (ND), and Primary Flight Display (PFD). Fortunately, SpectRM-RL provides the ability to represent control modes and displays.

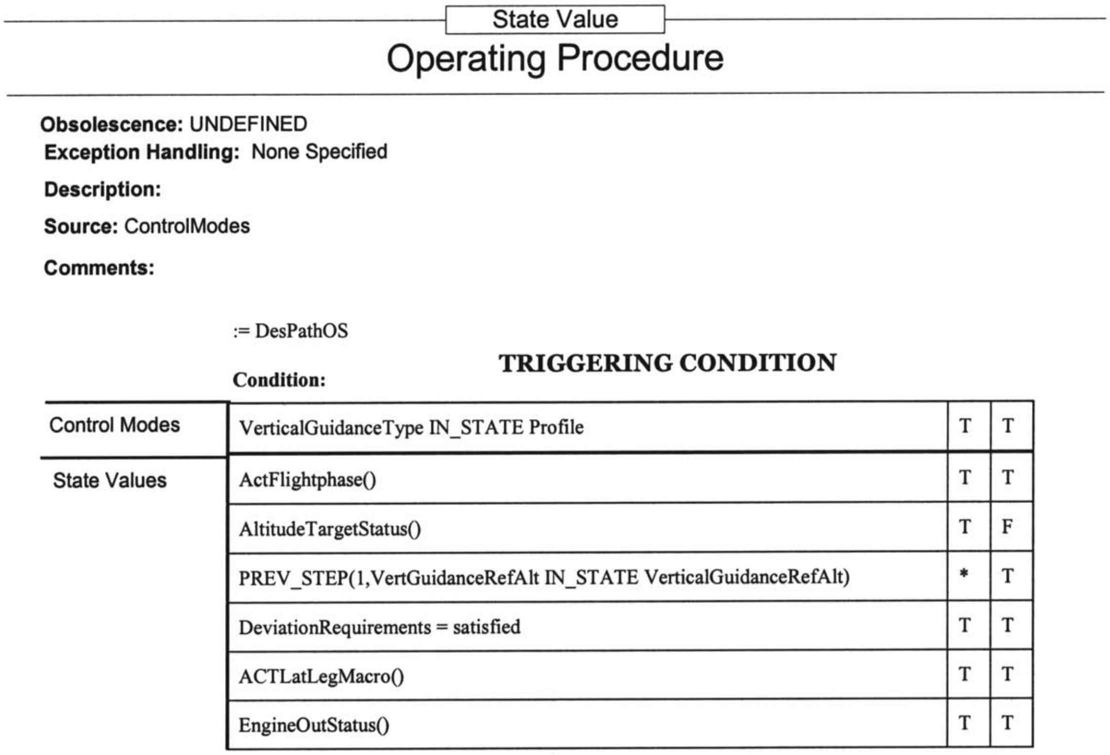
The descent phase will be used as an example because more accidents occur in this phase than others. A brief description of the descent phase is provided.

The descent phase normally starts when the aircraft departs the entered cruise flight level and lasts until the slats are lowered for landing. The descent vertical path is calculated by starting from the destination and computing in the reverse direction to satisfy decelerations, configuration changes, altitude and airspeed constraints, altitude speed limits, forecast winds, preselected descent speed, and other constraining factors. On a descent vertical path, the automation maintains continuous altitude control along a predefined path. When the aircraft speed exceeds the target speed by more than five knots, the automation expects the pilot to manually apply the speed brakes. An "ADD DRAG" message is displayed in the navigation display as a consequence.

However, if the airspeed continues to increase and exceeds the speed limit at an altitude 750 feet higher than the speed limit altitude, the FMS will automatically take action. It first displays the "SPEED LIMIT EXCEED" label in the Control Display Unit. It then replaces the target altitude by the speed limit altitude and stops the descent. The aircraft then is considered to be in a descent path overspeed scenario. This triggers a mode change in the automation from the Descent Path mode to the Descent Path Overspeed mode. The automation accordingly levels the aircraft, temporarily ending the descent. Once the airspeed becomes three knots less than the speed limit, the automation clears the message in the control display unit (CDU) and begins the descent again.

Amazingly enough the original system specification for the Descent Path Overspeed contains no reference to the supervisory control interface or to any type of display for that matter, except for the engagement of the profile as an operating procedure for the control modes.

Figure 3.3 shows the condition definition for the transition from any state to DescentPathOS for the Operating Modes state variable.



State Value

Operating Procedure

Obsolescence: UNDEFINED
Exception Handling: None Specified
Description:
Source: ControlModes
Comments:

:= DesPathOS

Condition:

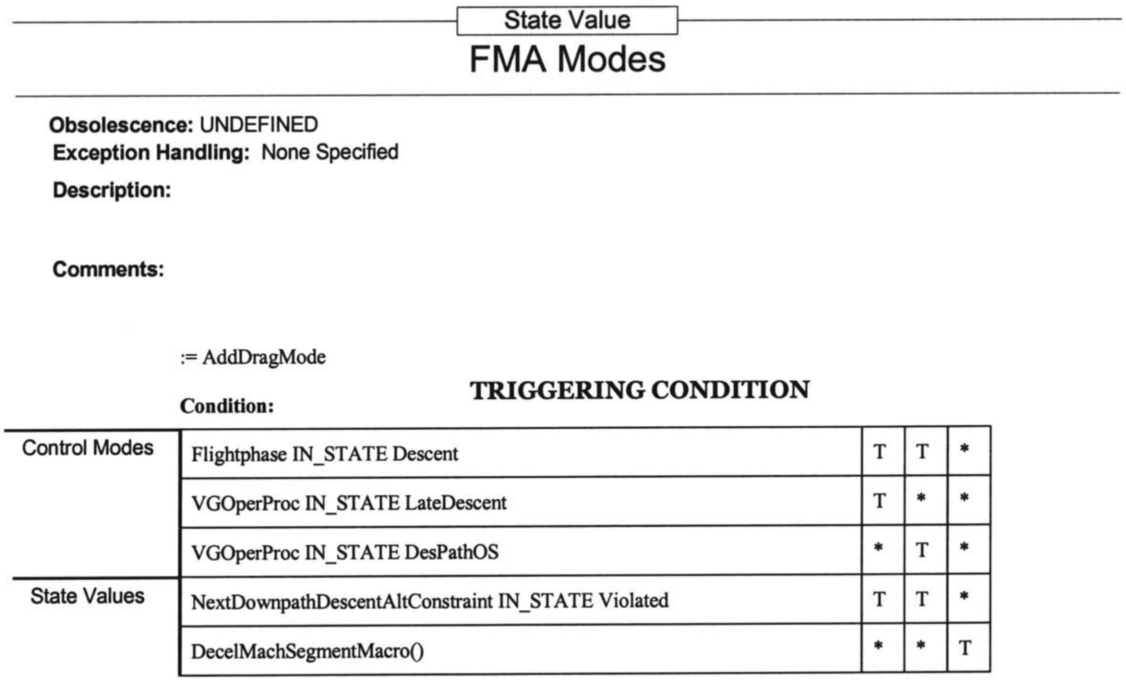
TRIGGERING CONDITION

Control Modes	VerticalGuidanceType IN_STATE Profile	T	T
State Values	ActFlightphase()	T	T
	AltitudeTargetStatus()	T	F
	PREV_STEP(1,VertGuidanceRefAlt IN_STATE VerticalGuidanceRefAlt)	*	T
	DeviationRequirements = satisfied	T	T
	ACTLatLegMacro()	T	T
	EngineOutStatus()	T	T

Figure 3.3. Ambiguous Interface Design

As shown in Figure 3.3 there is no description of the FMA or ND where this information is going to be displayed. The triggering condition refers to the active flightphase macro *ActFlightphase()* which defines a set of valid flightphases; the altitude target status macro *AltitudeTargetStatus()*, which defines the set of flight control computer engaged modes, and the PREV_STEP which is a built-in function – a function of the underlying formal RSM (Requirement State Machine) model – that allows referring to previous values of modes, state variables, outputs, and inputs. In this case it refers to the Vertical Reference Altitude input variable.

The display information for the add drag message appears as part of the specification of the FMA which corresponds to another different section in the model (Refer to section 2.2 for a list of the different sections of the model). Figure 3.4 shows the definition of the state variable that provides the add drag message. It is argued that these two state variables need to be combined in some way. They need to show at least that they are related to each other in the model. The indication of control modes and state variables – which is normally included as part of the output variable specification – can provide such indication. As a result, system designers can look at the Descent Path Over speed state variable and see what type of information is presented in the displays interface.



State Value

FMA Modes

Obsolescence: UNDEFINED
Exception Handling: None Specified
Description:

Comments:

:= AddDragMode

Condition:

TRIGGERING CONDITION

Control Modes	Flightphase IN_STATE Descent	T	T	*
	VGOperProc IN_STATE LateDescent	T	*	*
	VGOperProc IN_STATE DesPathOS	*	T	*
State Values	NextDownpathDescentAltConstraint IN_STATE Violated	T	T	*
	DecelMachSegmentMacro()	*	*	T

Figure 3.4. FMA Mode

Figure 3.4 shows how the add drag message is defined by the system specification. Note that there is no way to correlate the two state variables: FMA Modes with the Operating Procedure. Even though there is a reference to the vertical guidance operating procedure (VGOperProc), it is impossible to tell from the transition definition of Figure 3.3 that the operator is going to be informed about the transition. A simple algorithm can systematically detect this situation.

The SpecTRM-RL model is structured according to a RSM machine that contains its own data structure. There are a set of states, transition vectors, set of events, set of expressions, macros, constants, input variables, output variables, input receiver, and output receivers among others. It is possible then to analyze the model to the level of each event, condition, state, or table, and even target set according to the model of interest. Note that there can be more than one model running at the same time and communicating as is the case of the human task analysis and the system specification model.

Such a data structure combined with the mode classification scheme of SpecTRM-RL allows for the creation of interesting algorithms. For the example in question, we identified those transitions in the model that had inside their control mode any reference to the Operating Procedure state variable with the value of Descent Path Over Speed. An algorithm as follows can provide with helpful insight:

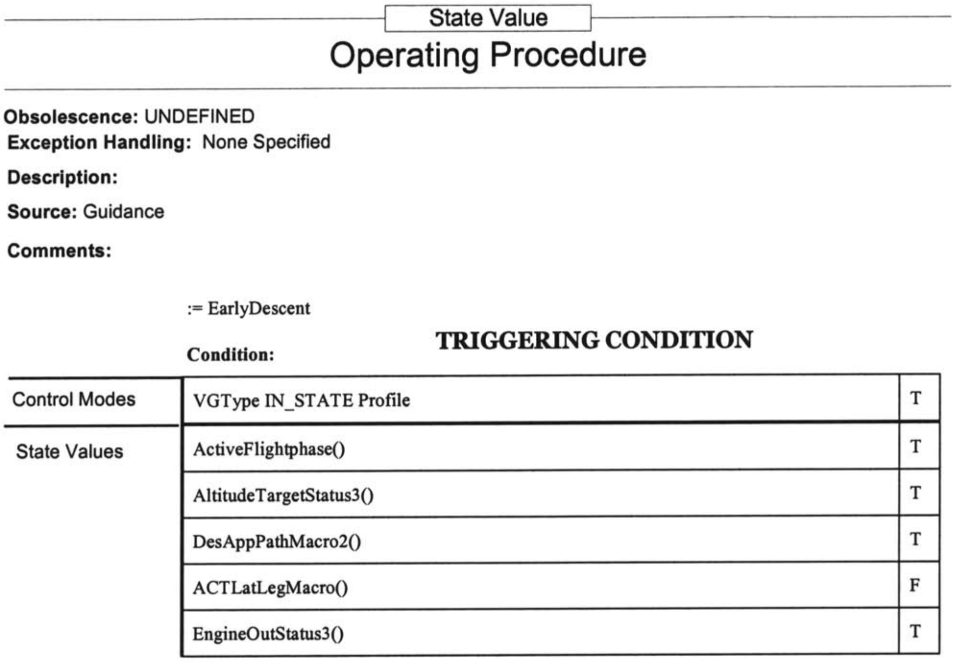
```
FOR every possible state
FOR every transition from the current state
FOR every column in the transition table
store all conditions specified in this column that relate to the CONTROL MODES
```

```
LOOP through the recorded conditions
    IF the condition definition includes the state variable [Operating Procedure] equals to [Descent Path Over Speed] THEN log it as part of an alert set
    ELSE do nothing
```

This pseudo code provides as a result the set of transitions and its conditions inside the AND/OR table that are reporting to the control modes the state of Descent Path Over Speed. This could have been used to discover how the FMA Modes was related to the state variable of interest.

Note also that a similar algorithm can be used to detect which transitions are not annunciated to the operator. This is the case of an indirect mode transition. In this case it will be easier because there will be those transitions with the CONTROL MODE set empty – following the same types of loops as discussed before.

Such types of algorithms can help designers visualize the different relationships of the variables used for the operating modes and the control modes thus assisting in the design of the interface. Note also that although the algorithm was designed by using computer science concepts such as loops and conditionals, the concepts itself were easy to visualize for non-computer literates. The transition is represented visually in the models by arrows signaling the beginning and end of the transition; the condition definition is part of the AND/OR table (basically the rows), the columns are also part of the table, and the control modes/state values are also present when system designers are creating these models since they have to select them. Therefore it is hypothesized that application domain experts without expertise in programming use and understand the tools and the analysis results.



State Value

Operating Procedure

Obsolescence: UNDEFINED
Exception Handling: None Specified
Description:
Source: Guidance
Comments:

:= EarlyDescent

Condition: **TRIGGERING CONDITION**

Control Modes	VGType IN_STATE Profile	T
State Values	ActiveFlightphase()	T
	AltitudeTargetStatus3()	T
	DesAppPathMacro2()	T
	ACTLatLegMacro()	F
	EngineOutStatus3()	T

Figure 3.5. Early Descent Definition

Another safety criteria discussed by Leveson relates to consistency of the interface. During the parallel development of the system model and the human task analysis inconsistencies were found corresponding to definitions of terms. An example where the same term was used with different connotation is provided. While comparing notes regarding the condition definitions for the transitions of the system specification to the human-task analysis, the development team came to the realization that for the definition of the Early Descent the engagement criteria for the vertical profile was different in the pilot's guide than the system specification.

Figure 3.5 describes the conditions defined in the system specification for the vertical flight control system regarding the engagement criteria for the Early Descent. In order for this operating procedure to be valid, the vertical guidance type should be in state profile. In contrast, the MD-11 Pilot's guide states that as soon as the pilot selects a descent altitude (different than cruise level flight) and selects a vertical speed, the vertical profile automatically disengages. By running the two models (or components as described by the data structure of SpecTRM-RL) a comparison can be made of when these two situations are different.

A simple algorithm can fill two arrays (one for each of the models) with the transition information, the fields (representing state variables, input variables, or macros), and the value for those fields. It will look only into those transitions with the desired value of the source/target for the transition thus minimizing the search space. It looks in this case for any transition with the word "EarlyDescent" as part of the transition definition. This is done for the two models: the human task analysis and the system specification.

The second part is to compare the two arrays. If the field and its corresponding value are different then it is recorded as part of the alert log of conditions that are in conflict. Otherwise, nothing happens since both models agree. For the inconsistency case the transition definition in the human task model is different from the system specification transition definition. For example, the system defines a transition from any state to EarlyDescent. In contrast, the human task analysis provides the conditions that transition from cruise level flight to early descent. The condition definitions of the transitions are also very different and this forces the algorithm to check only for the field name and its values. This assures that any transition with the Early Descent name and with same fields (state variables, inputs, or macros) but with different corresponding values, are identified to the system designer.

3.4.2 Conclusions

Based on a safety evaluation of the different ramification of the states, designers can create systems that provide more effective information. In addition, relationships that are hard to foresee for complex systems may be identified. The SpecTRM-RL language provides a framework in which the analyst can incorporate very helpful algorithms. Two examples were described that are related to design decisions regarding the interface. One of these design flaws was detected during a task analysis of the operations in the Pilot's Guide (Rodriguez et al., 2000). We believe that the visual manageability of SpecTRM-RL will allow manipulation of the data structure of the model by non-computer experts.

3.5 Operator Authority Limits

The purpose of this safety criteria is to create a well designed interlock. Such interlocks will prevent hazardous system states by enforcing correct sequencing of events or actions by isolating events in time. Design of such types of interlock features are crucial and delicate due to the problems that can arise in case a maneuver is prohibited that may be needed in extreme situations. There are many recent examples where pilots have been involved fighting with the automation over control of the aircraft after observing unexpected or undesirable behavior.

Important detection strategies could be concentrated on the natural sequences to accomplish a task and when input from the operator is ignored. An analysis can be conducted to determine if the blocking of a series of steps at a specific level will be counter-intuitive to the normal decisions of the user. In other words, there is a need to know where is the best place to input a forcing function and to know how to best inform users of this limitation.

The analysis is made easy by SpecTRM-RL because it provides the ability to run two models and do analysis on their interaction. It has been already described how a human task analysis can be formally represented in SpecTRM-RL in section 3.3. Therefore an algorithm will look at how these two models communicate to each other via the interfaces of the operator model and the system specification model. Conditions where the operator input are ignored are defined as:

All Actions from the Operator Model are considered as inputs to the System Specification if they are part of the communication points. For any of those transitions in which the condition definition does not include the actions of the operator model, then an alert is created providing the specific transition in the system specification where the condition occurs.

3.5.1 Conclusions

Operator authority limits analysis is an example of the ability of two models to allow analysis that otherwise would have been impossible. It is very hard to identify those conditions by only analyzing the system specification. A human task analysis can also be used to detect other error prone circumstances. Most of the analysis uses a combination of the RSM data structure and the information contained in the different models built used in this research.

4 Conclusions

This thesis was the result of an interest in developing safer software and the lessons learned from this effort. The research analyzed the process of designing software that will support operators in controlling complex systems. Software and system designers are in constant need of tools that will help them keep a manageable mental model of how complex systems work. Those tools help to create an abstraction of the system space to simplify it and reduce the potential for mode confusion and other types of errors. Unfortunately, such tools as formal methods have proven difficult to use by system designers, engineers, and human factors engineers. In this search for an easy to use tool, SpecTRM-RL proved to be a good tool for accomplishing those goals.

The first part of the thesis described the effort to build a complex system: the vertical flight control system for the MD-11 aircraft. The model included a human task analysis based on the pilot's guide. This task analysis was conducted by modifying a visual formalism previously developed by Brown and Leveson for doing ATC analysis. It proved useful not only in understanding how the system works but also in helping others understand the system and the pilot's tasks. We found that it was easier to refer to the model than to read to many pages of English text. It has the added benefit that the models built were translatable into SpecTRM-RL and therefore able to be systematically analyzed by existing algorithms to check completeness, consistency, and by the development of special purpose algorithms to detect specific types of errors.

The second part of the thesis explains how mode confusion analysis was performed in the models. SpecTRM-RL was designed to include mode specification.

Based on the underlying RSM data structure and the defined models of the system and human tasks, analysis algorithms were presented. Previous research in the area of mode confusion dealt only with after-the-fact conclusions about design flaws. This thesis showed that a priori and systematic analysis is possible with a language like SpecTRM-RL.

5 References

Advanced Technology Aircraft Safety Survey Report. Department of Transport and Regional Development.

Brown, M. and Leveson, N.G. (1998). "Modeling Controller Tasks for Safety Analysis." *Second International Workshop on Human Error, Safety, and System Development*, Seattle, April 1998.

Degani, A. Modeling Human-Machine Systems: On Modes, Error, and Patterns of Interaction. Ph.D. Thesis, Georgia Institute of Technology, 1996.

Degani, A., (1994). Modeling Human-Machine Errors: on Modes, Error and Patterns of Interaction. Doctoral Thesis. Atlanta, GA: Georgia Institute of Technology.

Dix, A., Finlay, J., Abowd, G., and Beale, R. (1993) Human Computer Interaction. Prentice Hall International, UK.

Fields, R., Wright, P., and Harrison, M.(1995) A Task centered Approach to Analyzing Human Error Tolerance Requirements. *Second International Symposium on Requirements Engineering* 1995.

Heindahl, M.P.E. and N. Leveson. Completeness and consistency analysis of statebased requirements. Transactions on Software Engineering, June 1996.

Javaux, D. (1998). "An Algorithmic Method For Predicting Pilot-Mode Interaction Difficulties." 17th Digital Avionics and Systems Conference. October 31 – November 6, Bellevue, WA.

Javaux, D. and De Keyser, V. (1998). "The cognitive complexity of Pilot-Mode Interaction. A Possible explanation of Sarter and Wood's classical results. In G. Boy & C. Graeber (eds.), Proceedings of the International Conference on Human-Computer Interaction in Aeronautics, May 27-29, Montreal, Canada.

Leveson, N. et all. (1997). "Analyzing Software Specifications for Mode Confusion Potential." Presented at the Workshop on Human Error and System Development, Glascow, March 1997.

Leveson, N.(2000) "Completeness in Formal Specification Language Design for Process Control Systems." To appear in the Proceedings of Formal Methods in Software Practice.

Leveson, N.G. and Palmer, E. Identifying Indirect Mode Transitions: 'Oops, it didn't arm' as a case study.

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

Leveson, N.G., M. Heimdahl, H. Hildreth, and J. Reese.(1994) Requirements specification for Process-control Systems. *IEEE Transactions on Software Engineering,* September, 1994.

Luttgen, G. and Carreño,V. (1999). "Analyzing Mode Confusion via Model Checking."

NASA/CR-1999-209332.

MD-11 Flight Management System Pilot's Guide. (1995). Honeywell Inc.

MD-11 FMC Software Requirements Document C72-3654-08. Honeywell Inc.

Miller, S.P., and Potts, J.N. (1999). "Detecting Mode Confusion Through formal modeling and Analysis." NASA/CR-1999-208971.

Nagel, D.C. (1988). Human Error in Aviation Operations. In Wiener, E.L., and Nagel, D.C. (eds.) Human Factors in Aviation. Academic Press, San Diego, California.

Norman, Donald. The Design of Everyday Things. New York, Basic Books. 1988.

Palmer, E. (1996) "Oops, it didn't arm" – a case study of two automation surprises. NASA technical report.

Palmer, E.(1996). "Oops, it didn't arm" – a Case Study of Two Automation Surprises. NASA Ames Technical Report.

Rodriguez, M. et al.(2000) "Identifying Mode Confusion Potential in Software Design". To appear in the 19th Digital Avionics and Systems Conference. Philadelphia, PA, 7-13 October.

Sarter, N. and Woods, D. (1995). Strong, Silent and 'Out-of-the-Loop': Properties of Advanced (Cockpit) Automation and their impact on Human-Automation Interaction, Cognitive Systems Engineering Laboratory. CESL Report 95-TR-01.

Sarter, N. D., Woods, D.D., and Billings, C.E. Automation Surprises. In G. Salvendy (Ed.) Handbook of Human Factors/Ergonomics. 2nd Edition, Wiley, New York.

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.

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

Vakil, S., Hansman, R.J. (1999). "Approaches to Mitigating Complexity-Driven Issues in Commercial Autoflight Systems." 3rd Workshop on Human Error, Safety, and System development. Liege, Belgium, June 7-8.

Vicente, K. and Rasmussen, J. (1992). Ecological Interface Design: Theoretical Foundations. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 4, July/August 1992.

Wiener, E.L. and Curry, R.E. (1980) Flight-deck automation: Promises and problems. NASA Technical Memorandum 81206, NASA Ames Research Center.

Zimmerman, M. et al.(2000) Making Formal Specifications Practical. To appear in the 19th Digital Avionics and Systems Conference. Philadelphia, PA, 7-13 October.