Final Report:

A DEMONSTRATION SAFETY ANALYSIS OF AIR TRAFFIC CONTROL SOFTWARE

Project Manager:
Prof. Nancy Leveson

Participants:

Prof. Earl Hunt (Psych), UW
Prof. Matt Jaffe (CS), ERAU
Dr. Susan Joslyn (Psych), UW
Prof. Nancy Leveson (CS), UW
Dr. Jon Reese (CS), UW
Prof. Alan Shaw (CS), UW
Prof. Zelda Zabinsky (IE), UW
Liliana Alfaro (IE), UW
Christine Alvarado (CS), Dartmouth
Molly Brown (CS), UW
Denise Pinnel (CS), UW
Jeffrey Samarziya (CS), UW
Sean Sandys (CS), UW
Michael Shafer (CS), UW

Preface and Acknowledgments

This report describes the results of a NASA contract to demonstrate our approach to safety analysis on a research version of the Center–TRACON Automation System at Dallas/Ft. Worth. A project of this magnitude and scope requires a multidisciplinary group contributing in varied ways. This is particularly true considering the short time frame of the project: The contract lasted six months although much of the detailed information about CTAS (for example, the system specification) was available only in the last six weeks. Because of the lack of detailed information about CTAS, this report should *not* be construed as saying anything about the safety of the current CTAS software.

We would like to acknowledge the following contributors to this project: Nancy Leveson was project manager and put the final report together. Sean Sandys built the SpecTRM-RL model with help from Michael Shafer and Jon Reese. Molly Brown created the task modeling language and the example handoff procedure with inputs from Nancy Leveson, Jon Reese, and members of the Flight Human Factors Research Group (led by Dr. Everett Palmer) at NASA Ames. The animations and visualizations were the product of Denise Pinnel and Christine Alvarado using a tool designed and implemented by Denise Pinnel. Alan Shaw examined the CTAS specification for timing issues. The preliminary hazard analysis and fault tree were done by Nancy Leveson with assistance from Matt Jaffe and Jeffrey Samarziya. The automated analyses were run by a team including Sean Sandys, Jon Reese, and Denise Pinnel. Nancy Leveson wrote the sections on mode-confusion analysis and intent specifications. Susan Joslyn conducted the human factors analyses with the assistance of Earl Hunt and input from Prof. Judy Ramey and Shuichi Koga. Liliana Alfaro and Zelda Zabinsky performed the operations research modeling and analyses. In addition, everyone provided input during project review meetings. We would also like to thank Prof. Earll Murman from the MIT Aero/Astro Dept. who attended the project meetings while on sabbatical at the University of Washington and provided useful advice.

This work was partially supported by grants from NASA Langley and NASA Ames.

Contents

1 Problem and Approach **1**

2 Standard Safety Analyses **8**
2.1 Preliminary Hazard Analysis 8
2.2 Fault Tree Analysis 11
2.3 Safety-Related ATC Requirements and Design Constraints 14
2.4 Hazard Analysis Techniques 15
2.4.1 System and Subsystem Hazard Analysis 17
2.4.2 Models 18
2.4.3 Search Techniques 19

3 Formal Model **22**
3.1 The SpecTRM-RL Modeling Language 22
3.2 The DFW TRACON Model 30
3.2.1 Overview of CTAS 30
3.2.2 What We Modeled 31
3.2.3 The FAST Model 32
3.2.4 Models of the Other System Components 37
3.2.5 Putting the Models Together 37
3.2.6 Handling Adaptation Data, STARS, etc. 37
3.2.7 Ease of Incorporating and Analyzing Additional Upgrades and System Changes 37

4 Controller Task Analysis **40**
4.1 Controller Task Modeling Language 40
4.2 Modeling Tasks Using SpecTRM-RL 44

5 Completeness and Consistency Analysis **49**
5.1 General Completeness Criteria 49
5.2 Timing Constraints 52
5.2.1 General Function and M&C Requirements 52
5.2.2 FAST Functions 55
5.2.3 TMC-GUI 55

6 Simulation and Animation **56**
6.1 Visualizations and SpecTRM-RL 57
6.2 Pseudo PVD 57
6.3 State Machine Model 59
6.4 Transition Tables 60

7 State Machine Hazard Analysis **63**

8 Deviation Analysis and FMECA **64**

9 Mode Confusion Analysis **70**
9.1 Mode Confusion 71
9.2 Interface Interpretation Errors 73
9.3 Inconsistent Behavior 75
9.4 Indirect Mode Changes 76
9.5 Operator Authority Limits 78
9.6 Unintended Side Effects 79
9.7 Lack of Appropriate Feedback 79

10 Human Factors Safety Analysis **81**
10.1 Human Error in the Current ATC System 81
10.1.1 Communication 82
10.1.2 Situation Awareness 84
10.1.3 Workload 85
10.1.4 Vigilance 86
10.2 Comparative Analysis of Present and FAST augmented ATC procedures . . 87
10.2.1 Before the Aircraft Enters the TRACON 88
10.2.2 Handoff from the Enroute Center 89
10.2.3 Sequencing and Spacing 90
10.2.4 Controlling Aircraft 91
10.2.5 Passive FAST 94
10.2.6 Summary of the Comparative Analysis 95
10.3 Outstanding Safety Questions 95
10.3.1 Decreased Situational Awareness? 95
10.3.2 Increased Vigilance Requirements? 98
10.3.3 Skills Degradation? 98
10.4 Experiments to Answer These Questions 99

11 Operations Research Modeling and Analyses **103**
11.1 Model 103
11.2 Scheduling Algorithms 105
11.3 Simulation Details 108
11.4 Results 110

12 Intent Specifications **113**
12.1 Goals for Intent Specifications 113
12.2 Intent or "Why" Abstraction 115
12.3 SpecTRM-RL Intent Specifications 116

13 Conclusions **119**

References **121**

A Minimum Separation Standards **127**

B Partial Fault Tree **130**

C Sample ATC Requirements and Constraints **138**
C.1 General ATC Safety Requirements 138
C.2 Automated System Requirements and Constraints 139

D Completeness Criteria for Black-Box Requirements Analysis **144**
D.1 General Considerations 144
D.2 State Completeness 145
D.3 Input and Output Variable Completeness 145
D.4 Trigger Event Completeness 146
D.4.1 Robustness Criteria 146
D.4.2 Nondeterminism 146
D.4.3 Value and Timing Assumptions 146
D.5 Output Specification Completeness 148
D.5.1 Environmental Capacity Considerations 149
D.5.2 Data Age 150
D.5.3 Latency 151
D.6 Output to Trigger Event Relationships 151
D.7 Specification of Transitions between States 152
D.7.1 Reachability 152
D.7.2 Recurrent Behavior 152
D.7.3 Reversibility 153
D.7.4 Preemption 153
D.7.5 Path Robustness 154
D.8 Constraint Analysis 155

E The SpecTRM-RL Models of CTAS at the DFW TRACON **157**

F Partial Intent Specification of TCAS **163**

1 Problem and Approach

Future changes to the air traffic control (ATC) system have two competing goals:

1. **Increase Throughput**: Requirements for moving both people and freight are increasing. A commonly stated goal is to triple the system throughput in the next 10 years. Whatever the specific numbers, the only effective way to accomplish the goal is to introduce more automation into the air traffic control (ATC) system.

2. **Decrease Accident Rate**: The number of aircraft accidents, especially those related ATC, is already very low. But given the current accident rate, it has been estimated that the projected increase in traffic will result in one major accident a week by the year 2005. NASA has stated a goal of reducing the accident rate by a factor of 5 in ten years and a factor of 10 in 20 years.

Automation is clearly involved in both of these goals, although perhaps in different ways. With the failure of the FAA's attempts to introduce big changes into ATC (e.g., the ill-fated AAS project), a new strategy was adopted that instead tries to incorporate smaller changes. CTAS (Center–TRACON Automation System) and passive FAST (Final Approach Spacing Tool), the focus of this study, are intended to assist the controllers in their tasks without introducing major changes in the way they do those tasks or in the ATC system in general.

CTAS has been very carefully defined and engineered. Our study found no significant safety implications in its use, although a few features deserve some additional consideration. Real throughput gains, however, and perhaps safety gains, are going to require the use of additional automation. Examples of proposed changes include free flight and direct communication between the ground ATC computer and aircraft flight management systems. In all these scenarios, air traffic controllers and pilots assume more of a monitoring role, although it is interesting that they are still tasked with the responsibility for safety without a clear shift of some of that responsibility to the automation where it will clearly lie.

Will safety increase or decrease with increased automation? Although theoretically software does not "fail" and computer failure rates are low, it is difficult (and perhaps impossible) to provide software that never does anything undesired or dangerous and to assure this property with high confidence. In addition, it is unlikely that humans will be entirely eliminated from the loop—humans will be needed to perform some functions that cannot currently be automated. It is in systems that straddle this middle ground—where humans and computers share control responsibility—that we find the most safety-related problems today.

Our very low ATC-related accident rates are the result of designing very large margins of error into the system. Aircraft are separated by relatively large distances and routed on jetways or paths that allow human monitoring and control as well as human error tolerance and recovery time. We cannot achieve the throughput goals (and others such as fuel efficiency), however, without reducing separation and spacing, which will reduce the tolerance for errors.

At the same time, traditional approaches to increasing safety and reliability developed for relatively simple systems are not as successful when applied to complex systems made up of electromechanical devices, computers, and humans. In simple systems, the majority of accidents are caused by single component failures: Increasing the integrity of components, therefore, is very effective in reducing losses. But in more complex systems, new types of accidents are arising that are often not the result of a simple combination of component failures. Perrow [Per84] calls these events *system accidents* and suggests that they are caused by interactive complexity in the presence of tight coupling.

High-technology systems, such as air traffic control, are usually made up of networks of closely related subsystems. Conditions leading to an accident emerge in the interfaces between subsystems, and coupling causes disturbances to progress from one component to another. System accident causes may be described as a coincidence of factors related to each other through a complex network and stemming from multiple independent events. Whereas in the past component failure was cited as the major factor in accidents, today more accidents result from dangerous design characteristics and interactions *among* components.

Computers have exacerbated the problems by allowing new levels of complexity and coupling with more integrated, multi-loop control in systems containing large numbers of dynamically interacting components. Increased complexity and coupling make it difficult for the designer to consider all the system hazards, or even the most important ones, or for the operators to handle all normal and abnormal situations and disturbances safely.

For the past 17 years, Professor Leveson and her graduate students have been developing a theoretical foundation for safety in complex, computer-based systems and building a methodology upon that foundation. The methodology (as described in her book *Safeware* [Lev95]) includes special management structures and procedures, system hazard analyses, software hazard analyses, requirements modeling and analysis for completeness and safety, special software design techniques including the design of human-machine interaction, verification, operational feedback, and change analysis.

The *Safeware* methodology is based on system safety techniques that are extended to deal with software and human error. We use automation to enhance our ability to cope with complex systems such as air traffic control. Identification, classification, and evaluation of hazards is done using modeling and analysis. To be effective, the models and analysis tools must consider the hardware, software, and human components in these systems. They also need to include a variety of analysis techniques and orthogonal approaches: There exists no single safety analysis or evaluation technique that can handle all aspects of complex systems. Applying only one or two may make us feel satisfied but will produce limited results. An effective safety program must span the entire life cycle and handle all potential accident causes.

We report here on a demonstration of the Safeware methodology to a research version of the Center-TRACON Automation System (CTAS) portion of the air traffic control system and procedures currently employed at the Dallas/Fort Worth (DFW) TRACON (Terminal Radar Approach Control). Although the Statement of Work asked only for a formal-methods-based functional correctness and completeness analysis tool and a tool for analyzing the system operations in the presence of failures, we felt that using only these tools would not provide an adequate assessment of safety and therefore demonstrated the additional activities necessary for a complete safety assessment.

Because safety analysis of a complex system is an interdisciplinary effort, our team included system engineers, software engineers, human factors experts, and cognitive psychologists. Figure 1 shows how our methodology and tools fit into a system engineering program. We did not include the usual safety activities that take place in the later parts of the life cycle due to the nature of the NASA AATT (Advanced Air Transportation Technologies) program, which focuses on concept development and system design. As we understand it, the FAA will be responsible for system implementation and operations.

We demonstrated most of the activities in Figure 1 although not necessarily as thoroughly as we might have given more time and information. Such a process is highly iterative and includes continual updating of what has been done previously as new information is gained through the system development process. In order to make the diagram less cluttered, the backward links are not shown, but note that the safety information system assists in this iteration process. An effective safety information system has been found to rank second only to top management concern about safety in discriminating between safe and unsafe companies matched on other variables [Kje87].

The parts of the process that we demonstrated for this project are:

1. **Preliminary Hazard Identification and Standard Hazard Analyses.** We were astounded to find that the Lincoln Labs CTAS system specification produced for the FAA not only does not include any information about safety but explicitly says that there are no safety requirements on the system. We performed a Preliminary Hazard Analysis (PHA) and defined some preliminary safety requirements and constraints for CTAS and for air traffic control in general. We also produced a partial fault tree for TRACON operations related to the operation of CTAS. The information we generated is used in the demonstration of our analysis techniques, and we discuss how it might be used for future upgrades.

2. **Modeling.** In order to do more than an evaluation of only the high-level ATC concept, a detailed specification or model of the behavior of the system components is required. A high-level design may appear to be safe while the detailed design contains hazardous component interactions. The hazards and design constraints identified in the first step must be traced to the system components, and assurance must be provided that the hazards have been eliminated or mitigated and the design constraints satisfied. Although theoretically this type of process could be performed on the detailed design of the system (including code if the component is a computer), a more practical approach is to provide hierarchical models and break the process up into steps. We built a state-based model using a language called SpecTRM-RL that

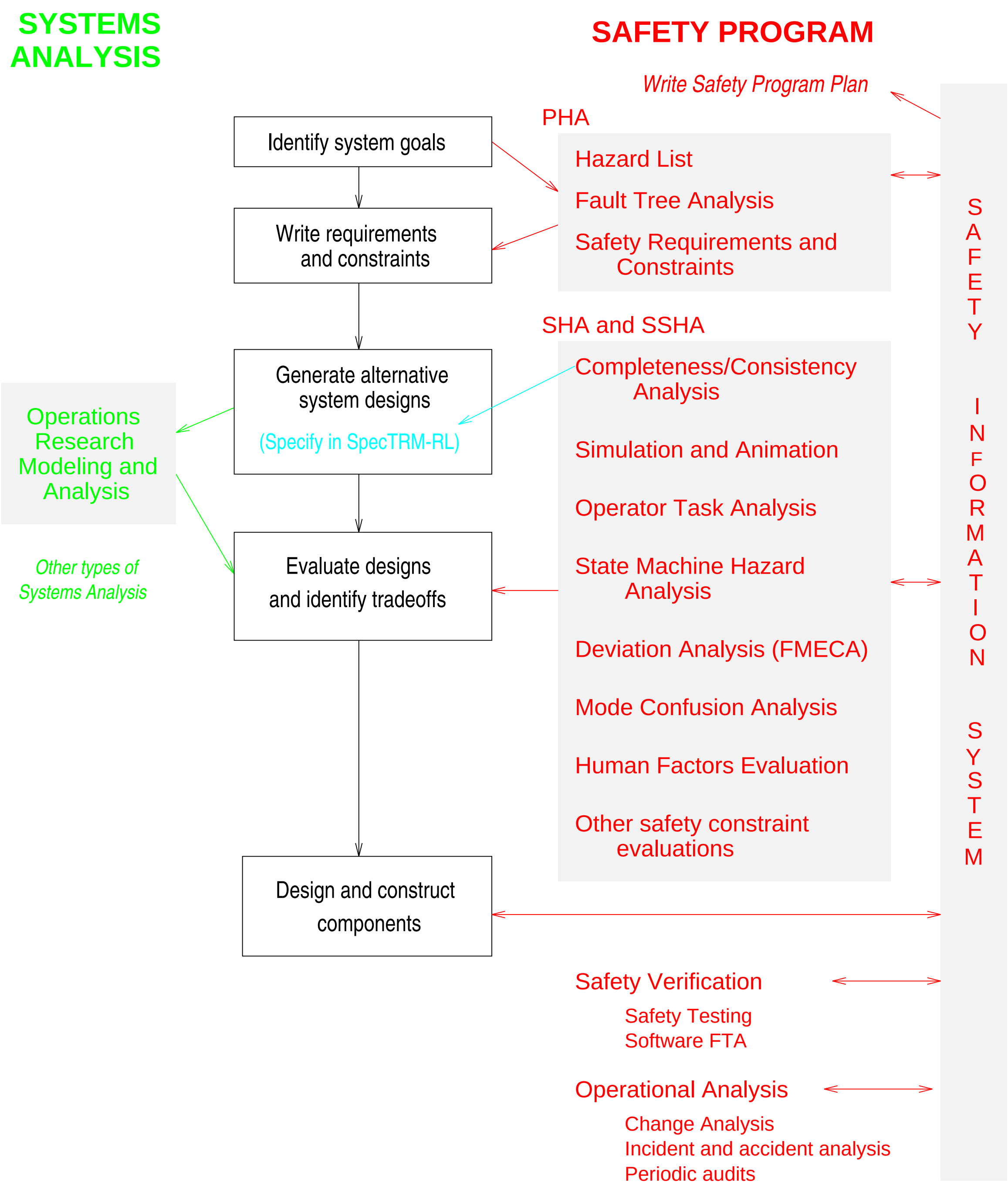


Figure 1: A sample safety engineering program.

is readable and understandable with minimal training but has a formal foundation that allows automated analysis. The models are also executable and visualization tools were built and used to animate the results of the model execution.

3. **Controller Task Analysis.** Humans form an important part of the ATC system, and they cannot be ignored in any safety analysis. In order to include operator procedures in our models, we devised a new modeling language with the same formal foundation as our SpecTRM-RL models. This model represents the nominal tasks that the controller (and sometimes the pilot) perform and can be analyzed and executed along with the SpecTRM-RL model of the other system components. We appreciate that humans do not necessarily perform tasks in the expected way. However, the first step is to determine whether the nominal or expected behavior is safe. The implications of human error or deviations from nominal behavior is investigated in our other analyses.

4. **Completeness and Consistency Analysis.** Accidents involving computers can usually be traced to incompleteness or other errors in the software requirements specification, not coding errors [Lut92, Lev95]. We have a set of formally-defined criteria to identify missing, incorrect, and ambiguous requirements in process-control specifications. These criteria include much more than the mathematical completeness that is checkable on most formal models, although we can check this too. Because the adaptation data for CTAS is part of our model, completeness and consistency of this data requires no extra effort or different techniques.

5. **Simulation and Animation.** Our models are executable and we have visualization tools to build animations appropriate to the model's domain (in this case, air traffic control). As an example, we have created an animation that shows the behavior of the aircraft within the TRACON area as the formal model is stepped through its states for a given set of inputs.

6. **State Machine Hazard Analysis.** Hazard analysis techniques that use backward search start with a hazardous state and determine the events that could lead to this state. The analysis starts from hazards identified during the preliminary hazard analysis and identifies their precursors. The information derived about both normal and failure behavior can be used to redesign the system to prevent or minimize the probability of the hazards.

7. **Deviation Analysis.** Forward search techniques start with an initiating event and trace it forward in time. Deviation analysis is a new type of forward hazard analysis technique we developed that is similar in its goals to HAZOP (HAZards and OPerability analysis), a very successful analysis procedure used in the chemical process industry. Deviation analysis allows reasoning about the effects of system parameter deviations on the components, that is, determining whether and how hazards can result from the system or components operating in an imperfect environment.

8. **Human Error Analysis.** Humans are and will continue to be an important part of any air traffic control system. Therefore, an effective safety program cannot just look at the automated parts of the system but must consider the impact of human error on the system and the effect of system design on human error. Increased automation in complex systems has led to changes in the human controller's role and, conversely, to new types of technology-induced human error. We approach this problem in two ways.

   The first is a method we are developing for using our formal system models to detect error-prone automation features early in the development process while significant changes can still be made. We have taken what has been learned from past accidents and simulator studies by cognitive psychologists and created a set of criteria to detect automation design features that are likely to induce human errors. The information produced from this *mode confusion analysis* can be used to redesign the automation to take out the error-inducing features or to design the human-machine interface, operator procedures, and training programs to minimize the errors.

   Our second approach to safety analysis of human error is more like classical human factors analysis. For this DFW CTAS study, we first looked at the types of human errors in the current ATC system and then performed a comparative analysis of the controller's job with and without CTAS. Normally this step would be followed by running experiments to determine the effect of the changes on human performance. However, the time limitations of this study did not allow us to perform this final step. Instead, we describe some relevant hypotheses and an experimental paradigm for evaluating these hypotheses.

9. **Operations Research Modeling and Analyses.** In addition to safety, air traffic control system designers must be concerned with efficiency. The systems engineering process involves making tradeoffs between various goals such as safety, throughput, and fuel efficiency. If a proposed upgrade turns out to degrade safety greatly while providing only minimal benefit in terms of throughput or fuel economy, then it may not be worthwhile to implement or an alternative design may provide a better result. We used a discrete-event simulation to compare the total delay and fuel burn for five different algorithms to schedule aircraft before they reach the TRACON feeder gates. The models can provide information such as the amount of delay or the amount of fuel consumed for various air traffic profiles operating under different scheduling algorithms. We show how these models can be used in tradeoff studies to evaluate proposed system designs.

10. **Intent Specifications.** The types of formal modeling and hazard analysis described so far provide a comprehensive assessment methodology. However, the most effective way to create a safe system is to build safety in from the beginning. The preliminary hazard analysis should start at the earliest concept formation stages of system development and the information derived from it and other safety analyses should

be used to guide the emerging design. One of the problems in achieving this goal is simply the difficulties inherent in communication among the diverse groups required to build a complex system. The goal of intent specifications is to provide a bridge between the various groups working on a system like air traffic control in order to ease coordinated design of components and interfaces and to provide seamless transitions and mappings between the various development and maintenance stages. The design rationale and the other information that is normally lost during development are preserved in a single, logically structured document whose design is based on fundamental principles of human problem-solving. Hazard controls and safety-related design constraints are traced from the highest levels down through system design, component design, and into hardware schematics or software code. We did not have the information necessary to build a complete intent specification for CTAS, but we describe what would be in it and show part of an example specification for TCAS II (Traffic Alert and Collision Avoidance System), which has similar aircraft tracking functions but is airborne and distributed on the individual aircraft. Our complete sample TCAS intent specification is too large (750 pages) to include in this report, but it can be viewed at the following URL: www.cs.washington.edu/research/projects/safety/www/intent/intent.ps

The next sections of the report describe each of these components of our methodology and their application to Build 2 of FAST (Final Approach Spacing Tool) operating in experimental mode at the DFW TRACON.

2 Standard Safety Analyses

A *safe* system is one that is free from accidents or unacceptable losses. Accidents result from hazards, where a *hazard* is defined as a system state or set of conditions that can lead to an accident (given certain other, probably uncontrollable or unpredictable environmental conditions). In safety engineering, any safety assessment starts with identifying and analyzing the system for hazards. Once the hazards are identified, steps can be taken to eliminate them, reduce their likelihood, or mitigate their effects.

In addition, some hazard *causes* can be identified and eliminated or controlled. Although it is usually impossible to anticipate all potential causes of hazards, obtaining more information about them usually allows greater protection to be provided with fewer tradeoffs, especially if the hazards are identified early in the design phase.

We demonstrated several standard safety analyses using the DFW TRACON as an example, including a preliminary hazard analysis (PHA), a fault tree analysis (FTA), and a failure modes and effects criticality analysis (FMECA). The FMECA is discussed later in the report in the section on Deviation Analysis (Section 8). The information we derived from the PHA and the FTA was used to specify some basic safety-related requirements and design constraints for ATC.

2.1 Preliminary Hazard Analysis

A PHA is used in the early life cycle stages to identify critical system functions and broad system hazards. It should be started early so that the information can be used in tradeoff studies and selection among design alternatives. However, this process is not done once and then considered to be complete: The process is iterative, with the PHA being updated as more information about the design is obtained and as changes are made. In addition, because the PHA starts at the concept formation stage of a project, little detail is usually available and early assessments of hazards and risk levels are necessarily qualitative and limited.

In general, performing a PHA involves:

1. Determining what hazards might exist during operation of the system and their relative magnitude.

2. Developing guidelines, specifications, requirements, and design constraints to be followed in system design.

3. Initiating actions for the control of particular hazards.

4. Identifying management and technical responsibilities for action and risk acceptance and assuring that effective control is exercised over the hazards.

5. Determining the magnitude and complexity of the safety problems in the program (how much management and engineering attention is required to minimize and control hazards).

The results of the PHA are used in developing system safety requirements, preparing performance and design specifications, evaluating the system design, test planning, preparing operational instructions, and management planning. In general, the results serve as a framework or baseline for later analyses and as a checklist to ensure that management and technical responsibilities for safety tasks are carried out. Doing a PHA is an absolutely critical step in any safety program for a complex system.

The PHA starts with the hazard list. Our first step was to determine if such a list already existed. We could find no list associated with the CTAS project so we contacted a system safety engineer at the FAA who told us he did not know of the existence of a hazard list for ATC. He said that such a list is not commonplace within the agency, although he is working hard to get hazard analysis requirements placed into new contracts. We looked at the System Specification for CTAS Build 2 written by MIT Lincoln Laboratory for the FAA. Not only is this specification devoid of any mention of safety requirements (even safety requirements not labeled as such but in the document anyway), but Section 3.7, **Safety Requirements**, contains only the following sentence:

> *There are no special safety requirements for CTAS beyond than* [sic] *the normal safety requirements for FAA equipment.*

In addition, Section 3.18 of the specification labeled **Precedence and Criticality Requirements** states that

> *All requirements specified in the document SHALL [4090] have equal weight.*

We checked again with the system safety office at the FAA to determine whether the "normal safety requirements" phrase in Section 3.7 held some important meaning or was simply irrelevant in this context and received the following reply:

> You're unfortunately correct [in assuming the phrase is irrelevant in this context], "normal safety requirements" are "2100" requirements like "no heavy lifting, no sharp corners, etc."

It was clear we were going to have to do a preliminary hazard analysis ourselves. We note that our PHA results have not been reviewed and thus should only be taken as an example. The PHA process normally involves intensive reviews by application experts and the government agencies involved. This review process has not been done for our PHA, and many changes would probably be required before it was acceptable for use with a real ATC upgrade. In addition, such a list would normally be continually updated with new hazards and information about previously identified hazards throughout system design, implementation, testing and evaluation, and operation. Therefore, our list should be considered as a starting point only.

One of the first steps in designing a safety-critical system, especially when the goal is to redesign parts of an existing system, is to establish a safety policy for evaluating upgrades. For example, how will design tradeoffs be made? Should they all be made with respect to possible changes in safety, even when those changes may be small in comparison to the other possible benefits of the change? Such a policy might be stated in terms of:

1. A possible goal accident rate or probability (usually stated in terms of 10 raised to some negative number) or

2. Changes to the nature or level of hazards where hazard level may be evaluated either qualitatively or quantitatively.

Using a goal accident rate makes decision making easy, but this rate usually is not predictable for a complex system in advance and can be determined only through extensive experience and use[1]. Particularly when the accident rate is already very low, determining whether it has changed is possible only over long periods of use. Use of the second type of policy requires more engineering judgment and review but may provide more accurate information upon which to base decisions. For example, a system change that resulted in only a small increase in fuel efficiency or throughput might not be considered worthwhile if it also required an increase in pilot–controller communication (which is known to be an important factor in ATC-related accidents).

In any case, the responsibility for establishing this policy rests with the FAA and not with the engineers designing upgrades for it. Such a policy may exist and we simply might not have come across it in the reading we have done.

The next step is to identify potential system hazards. In the hazard list, we included only the hazards related to TRACON operations and even then concentrated on automation aspects as this was the focus of the demonstration project. We identified six hazards[2]:

1. A pair of controlled aircraft violate minimum separation standards.

2. A controlled aircraft enters an unsafe atmospheric region (such as dangerous icing conditions, wind shear areas, thunderstorm cells).

3. A controlled aircraft enters restricted airspace without authorization.

4. A controlled aircraft gets too close to a fixed obstacle or terrain other than a safe point of touchdown on the assigned runway.

5. A controlled aircraft and an intruder (a VFR aircraft who should not be in controlled airspace) violate minimum separation standards.

    NOTE: Although the ATC controller is not strictly responsible for this airspace hazard, radar returns may be available that allow the controller to provide some prevention or mitigation of this condition.

[1] Arguments about the accuracy and usefulness of probabilistic risk assessment (PRA) in assessing the potential for accidents in automated systems where accidents may involve human, software, management, and organizational factors are complex and beyond the scope of this report.

[2] The number of hazards is usually small, even for a complex system. A list containing large numbers of hazards usually means that hazard causes have been included and not just the hazards themselves. There are important advantages to specifying a small high-level set of hazards first before attempting to identify what are usually a large number of hazard causes.

6. Loss of controlled flight or loss of airframe integrity.

The first step after identifying the hazards is to assess their criticality level (in ATC this level is almost always catastrophic) and to translate them into high level requirements and constraints on the system design. These requirements are refined as more information is obtained through the design and safety analysis programs. Some high-level requirements and constraints are described in Section 2.3 and an example set is shown in Appendix C.

2.2 Fault Tree Analysis

Fault trees are perhaps the most widely used system hazard analysis technique. We created a partial fault tree for the DFW TRACON. A complete fault tree would be very large, so we constructed the top levels of the tree for the hazards we identified in the PHA and showed the expansion of a few branches. Even following down only a few of the branches related to only one of the hazards, we generated about 150 boxes. Fortunately, the other branches will be very similar, with only small changes to reflect the particular hazard cause being evaluated. Figure 2 shows the highest level of the fault tree for the first hazard—loss of required separation between controlled aircraft (also called a *near-miss accident* or NMAC).

Figure 3 shows an example of a lower-level part of the fault tree for the NMAC hazard. An important question in any such exercise is when to stop expanding the nodes. Complex system fault trees containing only a few nodes will not be very useful. The more detail included in the fault tree and the more the nodes are refined, the more information will be available to the designers for eliminating or controlling the hazards.

Information from the fault tree can be used to generate safety-related system requirements and design constraints. Each of the leaf nodes in the tree (or a higher-level node that leads to that leaf node) should be traceable to a design characteristic that will eliminate or control it, or a decision must be made that the risk of that condition occurring will be accepted (examples are provided in the TCAS sample intent specification contained in Appendix F). In either event, the mitigation factor or the reason for risk acceptance must be documented.

Frequently, probabilities are attached to the leaf nodes of a fault tree and a probability calculated for the root hazard. Although this technique is very effective in systems made up of hardware components whose failure rates are well known, it is more problematic when applied to complex systems with software or human components or where the leaf nodes are not all "failures." In the attempt to get a calculated probability, sometimes accidents or hazards that do not result from simple component failures are omitted from such analyses in the interest of obtaining a probabilistic assessment. System accidents, which result from dangerous interactions between components rather component failures, are not included in such calculations and, in fact, are not well represented in the current fault tree framework.

There is controversy in the computer science community as to how failure rates for software should be determined or, indeed, whether such failure rates even make sense

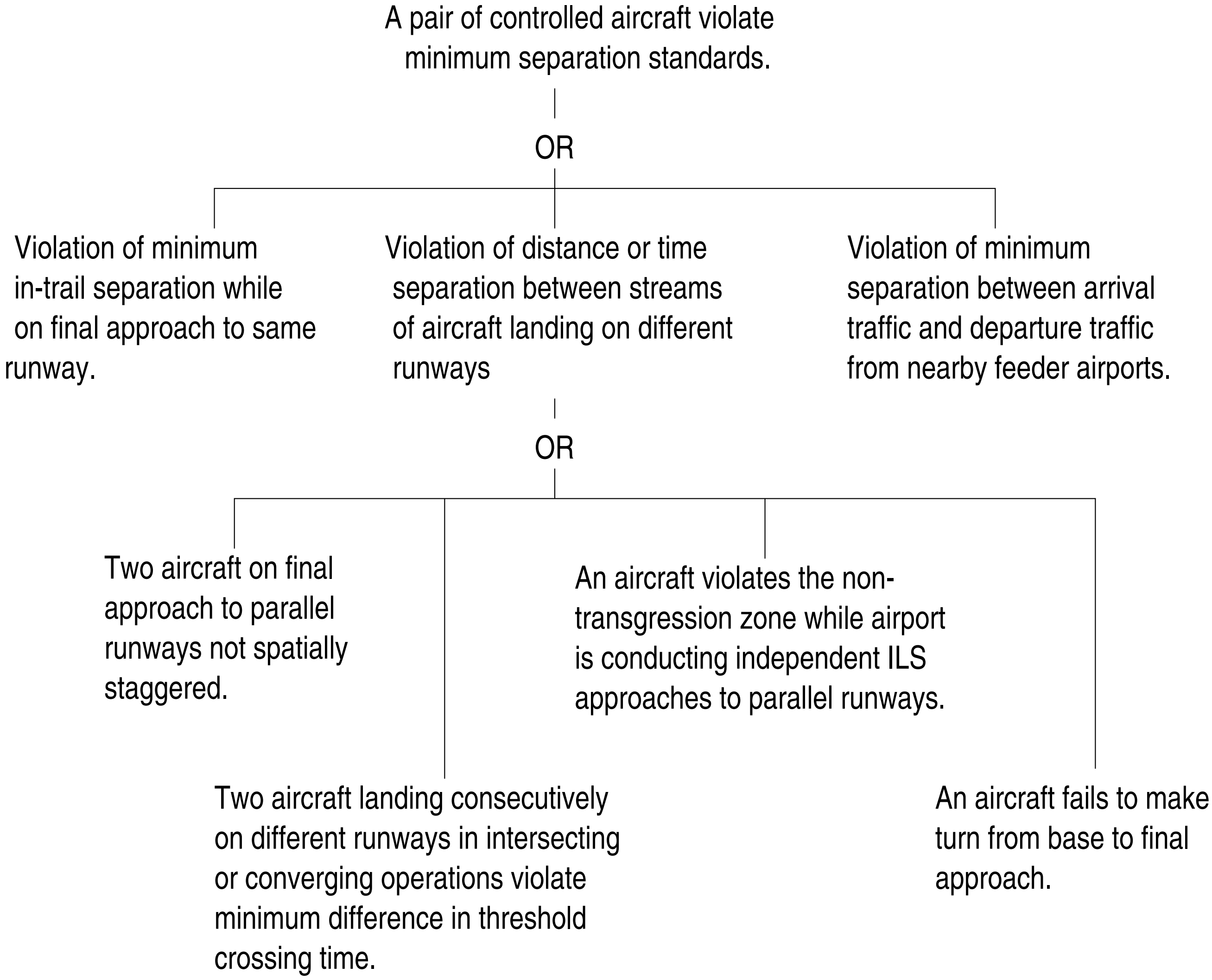


Figure 2: Highest level of the fault tree for loss of required separation between controlled aircraft

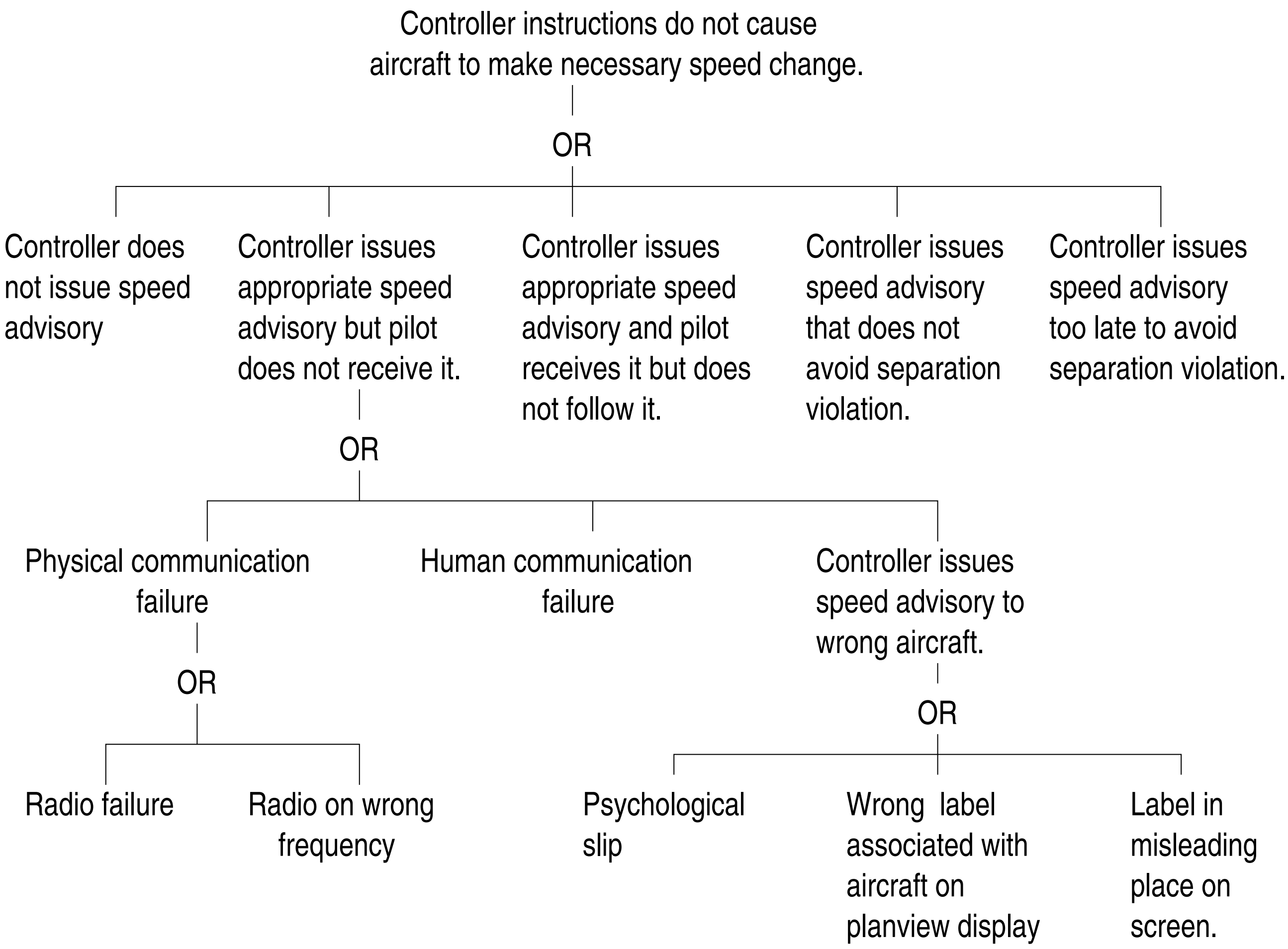


Figure 3: Lower levels of the fault tree for loss of required separation between controlled aircraft

for software, where errors are basic design flaws and the result of human mistakes rather than physical phenomena. Even if the basic concept is believed, the models proposed for measuring software reliability have not proven to be very accurate in practice. In addition, not all software behavior is hazardous so simply estimating the probability that software will do something that violates its specification is not a measure of whether it will do something dangerous. In fact, most accidents related to software arise because the specification was flawed, perhaps in failing to specify what to do in a particular case: A measurement of whether the software behavior complies with the specification is therefore unrelated to the potential contribution of the software to accidents.

Failure rates or probabilities are also difficult to obtain for humans controlling complex systems. In human reliability estimation techniques, human performance is viewed as a concatenation of standard actions and routines for which error characteristics can be specified and frequencies determined by observing similar activities in other settings. In such analyses, the task is modeled rather than the person. Rasmussen and others argue that such an approach may succeed when the rate of technological change is slow, but it is inadequate under the current conditions of rapid technological change [Ras87]. Computers and other modern technology are removing repetitive tasks from humans, leaving them with supervisory, diagnostic, and backup roles. Tasks can no longer be broken down into simple actions; humans are more often engaged in decision making and complex problem solving for which several different paths may lead to the same result. Only the goal serves as a reference point when judging the quality of performance—task sequence is flexible and very situation and person specific. Analysis, therefore, needs to be performed in terms of the cognitive information processing activities related to diagnosis, goal evaluation, priority setting, and planning. In addition, human actions cannot be separated from their context. All of these factors makes probabilistic analysis of human error very difficult and reduces confidence in the use of simple models.

We did not do a quantitative fault tree analysis. For the reasons stated and because of a lack of accurate probabilistic data for the leaf nodes, we felt it was not very useful in this instance. We note, however, that such analysis can be helpful in comparing designs, particularly those parts of a system where probabilistic data is available. Sometimes fault trees are abstracted to a few nodes (basically "hardware failure," "human error," and "software failure"), and failure rates are associated with those nodes, but we did not feel such an analysis would provide useful information. Another use of quantitative fault tree analysis is for sensitivity analysis, i.e., to determine which parts of the tree contribute the most to the hazard. This use of probabilistic analysis can be very helpful, but we did not have enough information, even about relative failure rates, to do a sensitivity analysis.

2.3 Safety-Related ATC Requirements and Design Constraints

Engineering safe systems involves not only evaluating risk but, even more important, includes designing ways to reduce it. Safety needs to be designed into a system; it cannot simply be measured or assessed into the system after the fact. There are usually other goals in any system design (for ATC these include increasing throughput and fuel efficiency), and tradeoffs are made continually during any design effort. By explicitly specifying the safety requirements and design constraints, tradeoff decisions can be made with respect to these requirements and constraints when alternative designs are being considered. In addition, there is no way to evaluate at the end of the design and implementation phases if the safety goals have been achieved if they have never been specified.

It is clear from examining CTAS and the papers written about it that the designers had a very clear idea of the safety requirements and constraints and did an excellent job of incorporating them into the CTAS design, but the system specification contains none of this information. A common problem in any long-lived system is that the designers do not specify why they made design decisions in a certain way, changes are made later that undo the careful design, and accidents result.

To specify ATC and CTAS requirements and constraints, we used the PHA and FTA results, experience we have had in specifying and performing safety assessments on similar systems, and our knowledge of causal factors in past ATC-related accidents or in related systems. Other types of information and processes that are helpful in this effort are described in Leveson's book *Safeware* [Lev95].

Figure 4 shows the first step in this process—associating each hazard with high-level requirements ("shall" statements) and design constraints ("must not" statements). Each of these high-level statements is then refined into more detailed requirements and constraints. We show a partial list in Appendix C.

Once again, the list in Appendix C should be considered only a sample of what types of requirements and constraints might be established for future ATC upgrades. Although in the past some have argued that automation providing information or advisories is not safety-critical because the controller ultimately has responsibility for safety and only the controller issues advisories, we find this argument unconvincing. If we hold the controller responsible for the safety of the airspace and the pilot responsible for the safety of the aircraft, then we must hold ATC system designers responsible for not interfering with the pilot's or controller's exercise of their responsibilities. If the design of tools causes or contributes to safety-critical human error, then that design is itself safety-critical. The most effective way to control the hazard may be to change the design of the tool and not to attempt to work around those flaws—especially when there is no independent way for the human operator to determine that the information being provided (on which human decisions are being made) is incorrect.

2.4 Hazard Analysis Techniques

Hazard analysis is the heart of system safety engineering. In general, two types of hazard analysis are usually performed: system hazard analysis and subsystem hazard analysis. Every hazard analysis involves some sort of search through the system design for hazardous states or conditions that could lead to hazards and a model upon which that search is based.

HAZARDS	REQUIREMENTS - CONSTRAINTS
1. A pair of controlled aircraft violate minimum separation standards.	1a. ATC shall provide advisories that maintain safe separation between aircraft. 1b. ATC shall provide conflict alerts.
2. A controlled aircraft enters an unsafe atmospheric region. (icing conditions, windshear areas, thunderstorm cells)	2a. ATC must not issue advisories that direct aircraft into areas with unsafe atmospheric conditions. 2b. ATC shall provide weather advisories and alerts to flight crews. 2c. ATC shall warn aircraft that enter an unsafe atmospheric region.
3. A controlled aircraft enters restricted airspace without authorization.	3a. ATC must not issue advisories that direct an aircraft into restricted airspace unless avoiding a greater hazard. 3b. ATC shall provide timely warnings to aircraft to prevent their incursion into restricted airspace.
4. A controlled aircraft gets too close to a fixed obstacle or terrain other than a safe point of touchdown on assigned runway.	4. ATC shall provide advisories that maintain safe separation between aircraft and terrain or physical obstacles.
5. A controlled aircraft and an intruder in controlled airspace violate minimum separation standards.	5. ATC shall provide alerts and advisories to avoid intruders if at all possible.
6. Loss of controlled flight or loss of airframe integrity.	6a. ATC must not issue advisories outside the safe performance envelope of the aircraft. 6b. ATC advisories must not distract or disrupt the crew from maintaining safety of flight. 6c. ATC must not issue advisories that the pilot or aircraft cannot fly or that degrade the continued safe flight of the aircraft. 6d. ATC must not provide advisories that cause an aircraft to fall below the standard glidepath or intersect it at the wrong place.

Figure 4: System hazards and associated requirements

2.4.1 System and Subsystem Hazard Analysis

System Hazard Analysis (SHA) usually begins as the design matures (around preliminary design review) and continues as the design is updated and changes are made. Its purpose is to recommend changes and controls, to evaluate design responses to safety requirements, and to document the corrective actions taken or controls instituted. Building on the PHA as a foundation, SHA involves detailed studies of possible hazards created in the interfaces between subsystems or by the system operating as a whole, including potential human errors. In a system like ATC where the design of most of the system already exists, it could start immediately.

SHA considers the system as a whole and identifies how the system operation, the interfaces between the system components, and the interfaces between the system and its operators can contribute to hazards. Specifically, SHA examines subsystem interfaces for:

1. Compliance with safety criteria in the system requirements specification;

2. Degradation of safety that could result from normal operation of the system and subsystems; and

3. Possible combinations of independent, dependent, and simultaneous hazardous events or failures, including erroneous behavior of safety controls and devices, that could lead to hazards.

SHA does not just consider the hardware components of a system. It can be used to identify high-risk human tasks and potentially safety-critical operator errors. This information can then be used in the design of the HMI and operational procedures to reduce errors and to provide facilities for operators to respond appropriately to automation failures and errors. Hazard analysis results should also be used in operator training to point out potential hazards and how they are controlled in the design or operational procedures. The goal is to help the operator understand and appreciate the potential consequences of bypassing protection features or inattention to safety-critical operations and to provide the information necessary for humans to respond appropriately in emergency conditions.

Several different techniques have been used in system hazard analyses. Some, like event trees, are appropriate only in the very specialized circumstances for which they were developed—in the case of Event Trees, for analyzing failures of nuclear power plant protection systems in response to a set of well-defined and single events (e.g., overheating of the core). Event trees are not appropriate for performing system hazard analyses on complex systems like air traffic control.

Subsystem Hazard Analysis (SSHA) examines individual subsystems and determines the effect of their operation or failure—including normal performance, operational degradation, functional failure, unintended function, and inadvertent function (proper function but at the wrong time or in the wrong order)—on system hazards. It also identifies necessary actions to determine how to eliminate or reduce the risk of identified hazards and evaluates the system design response to the safety requirements of the subsystem specification.

SSHA is started as soon as the subsystems are designed in sufficient detail, and it is updated as the design matures. As in SHA, design changes are evaluated to determine whether system safety is affected.

SHA and SSHA are accomplished in similar ways, but the goals are different. SSHA examines how *individual* component operation or failure affects the overall safety of the system, whereas SHA determines how normal and failure modes of the components *operating together* can affect system safety.

2.4.2 Models

Every hazard analysis requires some type of system model, which may range from a fuzzy idea in the analyst's mind to a complex and carefully specified mathematical model. The model may also range from a high-level abstraction to a low-level and detailed prototype. Nevertheless, information about the system must exist in some form, and that information constitutes the system model upon which the analysis is performed.

The specification (model) can be used to identify hazards or to analyze the system design for specific, known hazards. In either case, the model must be complete enough to provide the information necessary to achieve the analysis goals.

While a fuzzy model in the mind of the analyst may be adequate for preliminary hazard analyses and for simple systems, complex system hazard analysis requires a concrete model. To be cost-effective and useful for the many types of hazard analysis needed for these systems, the model should be usable by people from different engineering disciplines, perhaps assisted by automated tools and involving multiple views of the model. Building separate models for analyzing hardware, software, and human behavior will not only be expensive (perhaps prohibitively so), but disjoint models will not allow analyzing the interactions between these system components, which are where *system* accidents arise.

In addition, the modeling language should have a rigorously and unambiguously defined semantics and be readable by application experts and the user community. If the specification and analysis results are not readable and reviewable by system safety and application experts, confidence in the results will be lessened. Readability and reviewability will be enhanced by using languages that allow building models that are semantically close to the user's mental model of the system. That is, the *semantic distance* between the model in the expert's mind and the specification model should be minimized.

Furthermore, the language must be both formally analyzable and readable without advanced mathematical training. Ideally, the specification language should reflect the way that engineers and application experts think about the system, not the way mathematicians do. While automated tools are helpful and may even be necessary to analyze some aspects of large and complex models, we believe (and our empirical evidence [MLRPS97] and industrial experiences support this view) that the most important errors will be found by expert (human) review. The analysis tools we build and the mathematical theories upon which they are based cannot possibly incorporate all the domain-specific knowledge such as FAA rules and procedures, basic aeronautical engineering, human factors and cognitive psychology, and so on required to find subtle safety-critical flaws in a complex system design. On the other hand, the complexity of these systems leads to the necessity to use formal models and automated assistance to support human navigation and understanding of the models and system specifications and to perform automated analysis where possible. We solve this dilemma by providing reviewable specifications based on an underlying formal model.

Another reason for the need to provide reviewable modeling languages is that any potential design flaws detected by automated tools will need to be evaluated by humans. Thus readability of the models is a requirements for human processing of the analysis results.

Finally, the economics of system development are unlikely to allow building multiple models or specifications. Instead, we have devised a way—which we call intent specifications—to integrate formal and informal specifications so that the analysis tools work directly on the normal system specifications (see Section 12).

2.4.3 Search Techniques

A safety analysis involves a search of the model. How the search is performed depends on the structure of the model and the goal of the search. One classification for such search techniques is forward or backward while a second is top-down and bottom-up [Lev95].

A *forward* (sometimes called *inductive*) search takes an initiating event or condition and traces it forward in time. The result is a set of states or conditions that represent the effects of the initiating event. An example of such a search is determining how the loss of a particular control surface will affect the flight of an aircraft or how the garbling of some radar data will affect the tracking of an aircraft in an air traffic control system.

Tracing an event forward can generate a large number of states, and the problem of identifying all reachable states from an initial state may be unsolvable using a realistic amount of resources. For this reason, forward analysis is often limited to only a small set of temporally ordered events.

In a *backward* (also called *deductive*) search, the analyst starts with a final event or state and identifies the preceding events or states. This type of search can be likened to Sherlock Holmes reconstructing the events that led up to a crime. Backward search approaches are useful in accident investigations and also in eliminating or controlling hazards during system development by, in essence, investigating potential accidents before they occur.

The results of forward and backward searches are not necessarily the same (see Figure 5). Tracing an initial event forward will most likely result in several final states, not all of which represent hazards or accidents. Because most accidents are caused by multiple events, to be fully effective the forward analysis must include more than a single initiating event. Combinatorial explosion usually makes an exhaustive search of this type impractical and limits the number of initiating events that can be considered. The advantage of a forward search is that hazards that have not previously been identified can theoretically be found.

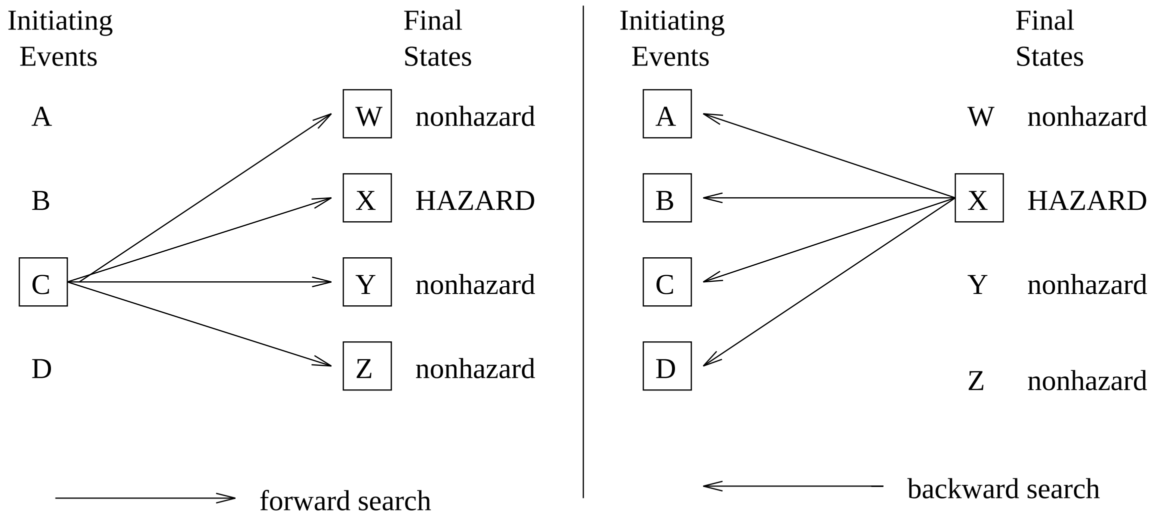


Figure 5: Forward search and backward search provide different types of information. In particular, the states found in the two searches will probably not be the same.

Tracing backward from a particular hazard or accident to its preceding states or events may uncover multiple initiating or contributing events, but the hazards must be identified. System engineers are quite effective in identifying system hazards, of which there are usually a limited number. Finding all the causes of such hazards is a much more difficult problem. It is easy to see that if the goal is to explore the precursors of a specific hazard or accident, the most efficient method is a backward search procedure. On the other hand, if the goal is to determine the effects of a specific event, a forward search is most efficient.

A second classification for search techniques is top-down and bottom-up [Lev95]. Here the relationship being investigated is structural (whole–part): Higher-level abstractions are refined or broken down into their constituent parts. In a top-down search, a basic event, set, task, or system is broken down into more basic events, conditions, tasks, or subsystems. When the search is bottom-up, subcomponents are put together in different ways to determine the result. An example of a top-down search is identification of all the ways two aircraft could violate minimum separation requirements. A bottom-up search might examine the effect of a particular incorrect output from FAST.

As with forward and backward searches, the results of top-down and bottom-up searches are not the same. For example, examining only the effects of individual component failures on the overall system behavior (a bottom-up search) misses hazardous system behavior that results from *combinations* of subsystem failures or from the interactions among nonfailure (correct) behavior of several subsystems.

As in forward searches, considering the effects at the system level of all possible combinations of component behavior using a bottom-up approach is usually not practical. Top-down searches that start from a hazardous system state will in most cases be more useful in achieving this goal. On the other hand, determining the effect of a particular component failure on system behavior is, theoretically, most efficiently accomplished using a bottom-up search. Accomplishing this latter goal, however, is often very difficult for complex systems.

Some search strategies do not fit into one of these categories, such as cause-consequence analysis. Instead, the search starts with some event or deviation and uses a combination of forward/backward and top-down/bottom-up searches to find paths between hazards and their causes or effects. Cause-consequence analysis, for example, combines forward and top-down searching. In any of the techniques, the search may start with deviations, failures, changes, and so on.

In a previous experiment, we applied several different types of hazard analysis techniques to the specification of experimental NASA Ames guidance software for a high-speed civil aircraft using our formal specifications and automated tools. We found that each analysis detected different types of errors and hazards. Thus, together the techniques provided a more comprehensive safety analysis than any individual technique [MLRPS97]. We also discovered that the more the analysts knew about the application and the better they understood the model, the more successful they were in finding potential problems.

In the rest of this report, we describe our modeling language and the analyses we believe should be performed in building safety-critical complex systems. We provide examples using passive FAST in the DFW TRACON environment as the test case, although the limited nature of passive FAST does not allow for the most effective demonstration of the usefulness of sophisticated analysis tools.

3 Formal Model

As stated earlier, all hazard analysis starts from some type of model. We designed a modeling language, called RSML (Requirements State Machine Language), to specify the system requirements for TCAS II, an airborne collision avoidance system required on most aircraft in U.S. airspace [LHHR94]. Our TCAS II specification and the RSML language were adopted as the basis for the official FAA TCAS specification, and RSML is still being used to specify changes and upgrades to the system.

In the process of formally specifying the system requirements for TCAS II, we learned a lot about how to design a reviewable specification language—our specification was reviewed by a large number of people having varied backgrounds and knowledge, such as computer scientists, aeronautical engineers, and pilots. We also learned what features appear to be the most error-prone in such a language.

Our goals in designing an improved language, which we call SpecTRM-RL, include enhancing readability, eliminating or changing features (such as internal broadcast events) we found to be especially error prone in use, providing more support for building blackbox models (specifiers who are used to including internal design in their specifications seem to have difficulty building pure blackbox models), enforcing certain constraints to prevent design features that are known to lead to accidents, and enhancing the ability to manually check for these features.

SpecTRM-RL is at the heart of a CAD system (system engineering workbench) Leveson and colleagues are building called SpecTRM (Specification Tools and Requirements Methodology). Some of the SpecTRM tools and analysis techniques are described and demonstrated in this report. We first describe the modeling language and then the analysis techniques and tools.

3.1 The SpecTRM-RL Modeling Language

Our models describe a system in terms of the blackbox behavior of the components. A blackbox model of behavior permits statements and observations to be made only in terms of outputs and the inputs that stimulate or trigger those outputs. The model does not include any information about the internal design of the component itself, only its externally visible behavior. The overall system behavior is described by the combined behavior of the components, and the system design is modeled in terms of these component behavior models and the interactions and interfaces between the components.

In order to make the language formal enough to be analyzable and yet readable and reviewable by non-mathematicians, we have defined a formal model that underlies a more readable specification language(s). The underlying formal model is independent of any specific, existing requirements language—we call it a requirements state machine (RSM). It is based on a Mealy automaton, which provides a convenient abstraction for most state-based specification languages.

For readers unfamiliar with such models, a state machine model shows the states of the

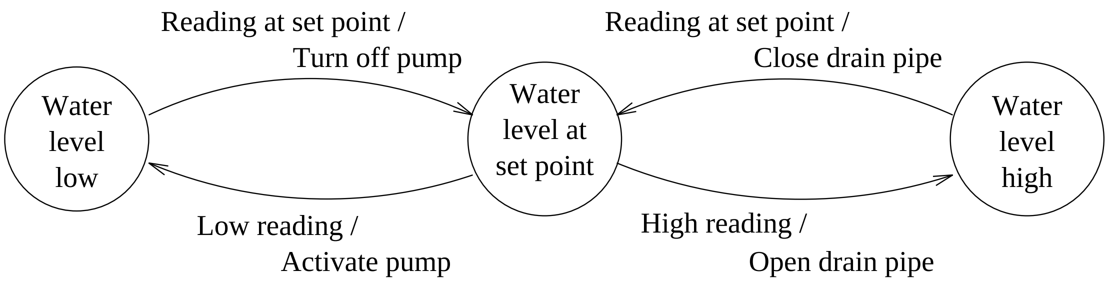


Figure 6: A state machine model of a water level control.

system and the events that cause state changes (transitions between states). For example, Figure 6 shows a simple model of a water-level controller for a water tank.

This model has three states (represented by circles): water-level-low, water-level-high, and water level at the set point. The directed arcs represent transitions between states. Each arc shows the condition for changing state (e.g., the water level reading is too high) and an action to be taken when the transition occurs (e.g., open the drain pipe). When the machine is in a particular state and the conditions on a transition from that state become true, the machine changes to the new state and implements the action. In the example, depending on the sensor reading of the water level and the current state of the model, the controller will activate the pump, turn off the pump, open the drain, or close the drain.

Finite state machines are most appropriate for modeling digital devices with a finite (although perhaps extremely large) number of states. But a finite-state model of an analog or continuous device can also be useful if we are interested only in discrete categories of the device's behavior. In the water controller example, the required behavior of the controller can usefully be specified by only three water level states into which the entire state space can be divided. Most safety analyses fall into this category.

One problem with finite-state models of complex systems is that the large number of states can make writing the model down impossible. We estimate that our model of TCAS II has at least 10^{60} states, and our CTAS model has a similar number. To solve this problem, we use a metamodel from which the entire state space could theoretically be generated. The state space is, of course, never actually generated; the analysis is instead performed on the metamodel itself. The metamodels we have constructed for even very complex systems have required only a few hundred state variables.

Our models rest on the concept of a basic control loop (see Figure 7). The controller reads the sensor data and using this information, along with perhaps other information, formulates and issues a command to an actuator that actually manipulates the process in some way to achieve the overall goals while satisfying constraints on the way those goals can be achieved. Although FAST and the other CTAS components do not themselves issue control commands, they provide the information (and sometimes the control command itself) to the human controller who acts partly as a controller and partly as an actuator

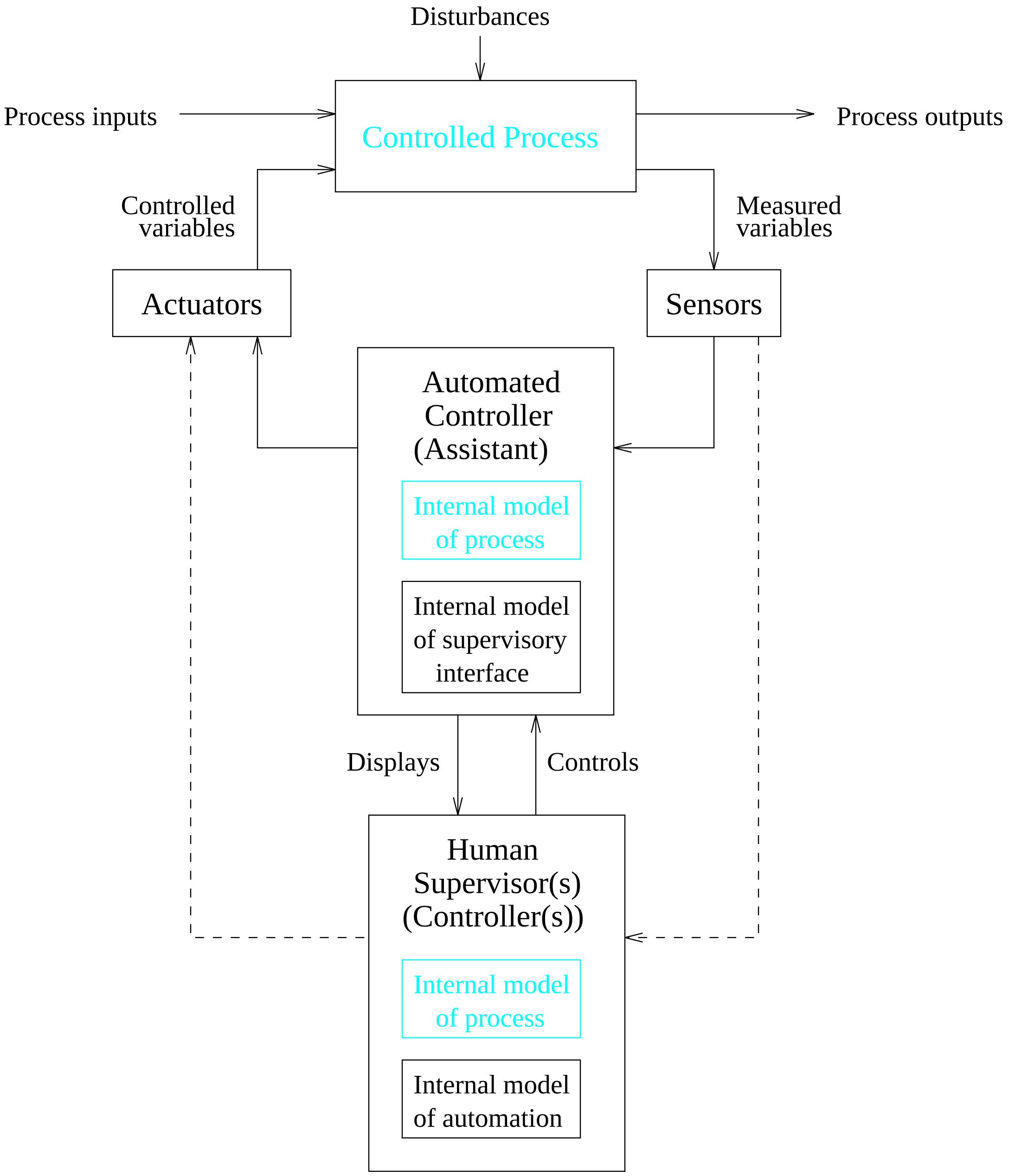


Figure 7: A basic control loop. A black-box requirements specification captures the controller's internal model of the process. Accidents occur when the internal model does not accurately reflect the state of the controlled process.

for the CTAS software (that is, in FAST the direct link shown in Figure 7 from the automated controller to the actuators is missing). Therefore, the CTAS components of the ATC system are acting de facto as controllers and it is appropriate to model them as control software.

All control software (and any controller in general) uses an internal model of the general behavior and current state of the process that it is controlling. This internal model may range from a very simple model including only a few variables to a much more complex model. The model may be embedded in the control logic of an automated controller or in the mental model of a human controller and is used to determine what control actions are needed. The model is updated and kept consistent with the actual system state through various forms of feedback.

When the controller's model of the system diverges from the actual system state, erroneous control commands (based on the incorrect model) can lead to an accident [Lev95]—for example, the software does not know that the plane is on the ground and raises the landing gear or it does not identify an object as friendly and shoots a missile at it. The situation becomes more complicated when there are multiple controllers (both human and automated) because the system models of the various controllers must also be kept consistent. In addition, pilots or air traffic controllers who are supervising or using automated assistance must not only have valid models of the aircraft and the controlled airspace (respectively), but they must also have a model of the automated systems' behavior in order to monitor or control the automation as well as the aircraft or airspace.

The automated controller also has a model of its interface to the human controllers or its supervisor(s). This interface, which contains the controls, displays, alarm annunciators, etc., is important because it is the means by which the two controllers' models are synchronized.

We represent the controlled process and supervisory interface models using state machines and define required behavior in terms of transitions in this machine. The goal of hazard analysis using our models is to ensure that the specified operation of the component (i.e., the model of the component's required behavior) will not lead to hazardous states in the controlled process.

State space explosion is prevented in our models by dividing a component into subcomponents and specifying the states of each of these parallel components as separate state machines. The complete model then becomes the cross product of these state machines. For example, Figure 8 shows two parallel state machines that might be part of an ATC model. One describes the current traffic density—low, average, and high—while the second represents up to 90 schedule slots. Each of the schedule slots is either available, currently scheduled, or blocked by the controller (i.e., is not presently available for scheduling by CTAS). The current modeled state of the system (if these are the only two state machines in the model) is a combination of the traffic density and the state of each of the 90 schedule slots. In general, the entire possible state space is the cross product of the parallel state machines or, in other words (for this case), every possible combination of traffic density level and schedule slot availability.

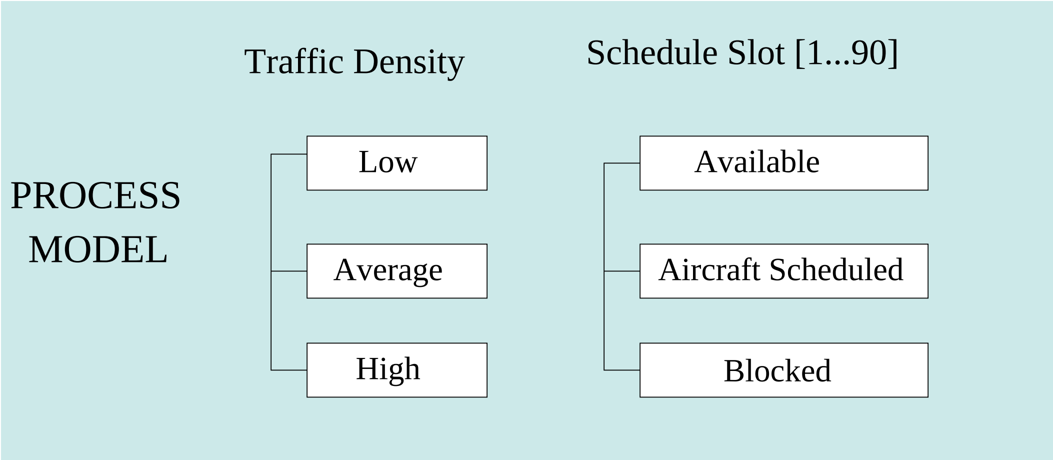


Figure 8: An example of how the *process model* is modeled using parallel components to avoid the problem of state explosion

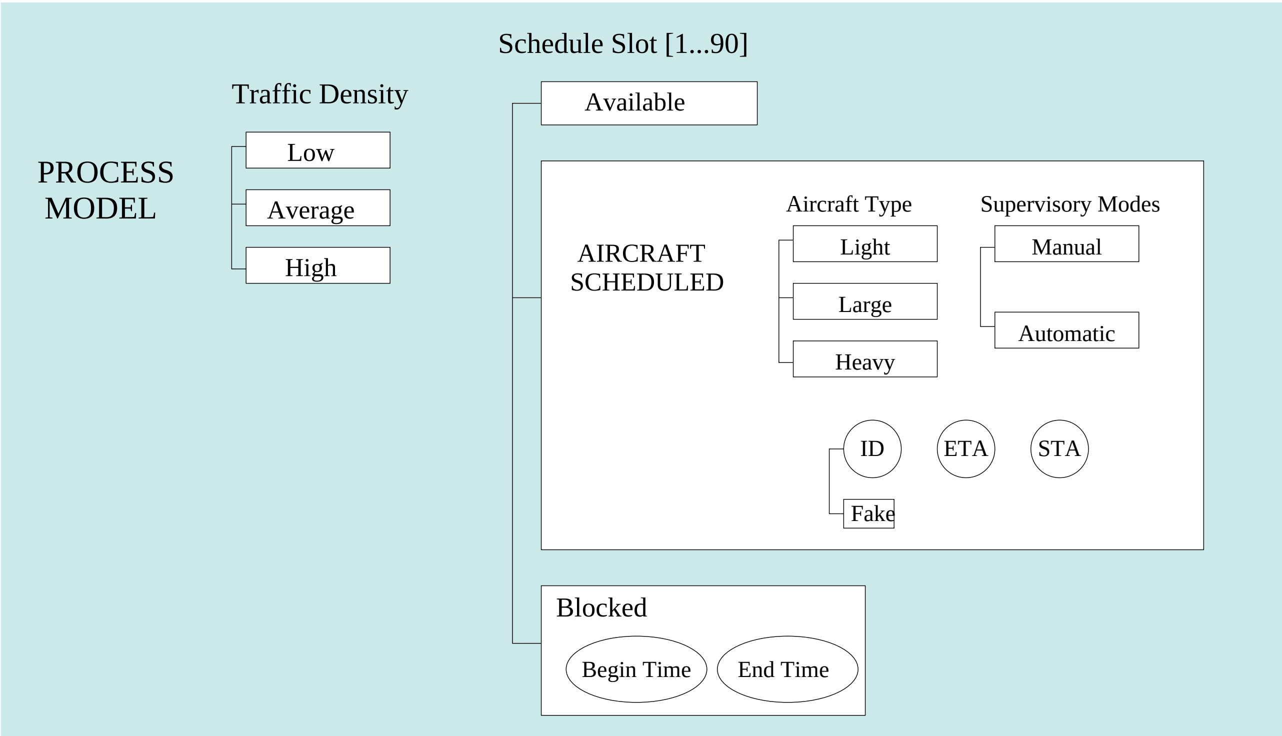


Figure 9: An example of how the *process model* parallel components can be refined to show the properties of the components

Each state may be further refined into additional parallel state machines. For example, the states *Aircraft Scheduled* and *Blocked* can be further refined to show properties of these states as shown in Figure 9. Thus, if a slot is scheduled, then the schedule slot state also contains the aircraft type (light, heavy, large), ID, estimated time of arrival (ETA), scheduled time of arrival (STA), and supervisory mode (manual or automatic). If a schedule slot has been blocked by the controller (perhaps to handle emergency traffic or popups), the blocked begin and end time will be represented in the state.

Our past experience in using these types of models has demonstrated that too many levels of this type of refinement abstraction is more harmful than helpful in terms of human understanding of the models. Therefore, we try to minimize the number of levels in the model.

A SpecTRM-RL model of a control component has three parts: (1) a specification of the operating modes of the component, (2) a specification of the component's view of its interface with the other components, and (3) a model of the controlled process.

The first part of a SpecTRM-RL model is a specification of the operating modes of the component, for example, waypoint-capture mode or route-intercept mode. An operating mode defines a mutually exclusive set of system behaviors. For example, the following table shows the possible transitions between states in a simple state machine given two system modes: startup mode and normal operation mode:

	a	b	c	d	e
Startup	c	b	d	e	a
Normal	c	d	a	e	a

Table 1: A simple state machine with two modes

The startup and normal processing modes in this machine determine how the machine will behave. If the conditions occur that trigger a transition from state c, for example, the machine will transfer to state d if it is in startup mode or to state a if it is in normal processing mode. Note that this table does not show all the conditions under which the transitions may be triggered, only the operating modes. In general, state transitions may be triggered by events, conditions, or simply the passage of time. The current operating mode determines how these triggers will be interpreted and what transitions will be taken.

The concepts of state and mode are not differentiated in our underlying formal RSM models—they are both represented as states—but the concept of a mode is a useful abstraction in describing the behavior of control systems. Engineers often refer to modes when describing required system functionality, and mode confusion is frequently implicated in the analysis of operator errors that lead to accidents (see Section 9). Therefore, SpecTRM-RL includes a mode abstraction and our analysis techniques incorporate the concept of modes.

Figure 10 shows the operating modes part of our TMA (Traffic Management Advisor)

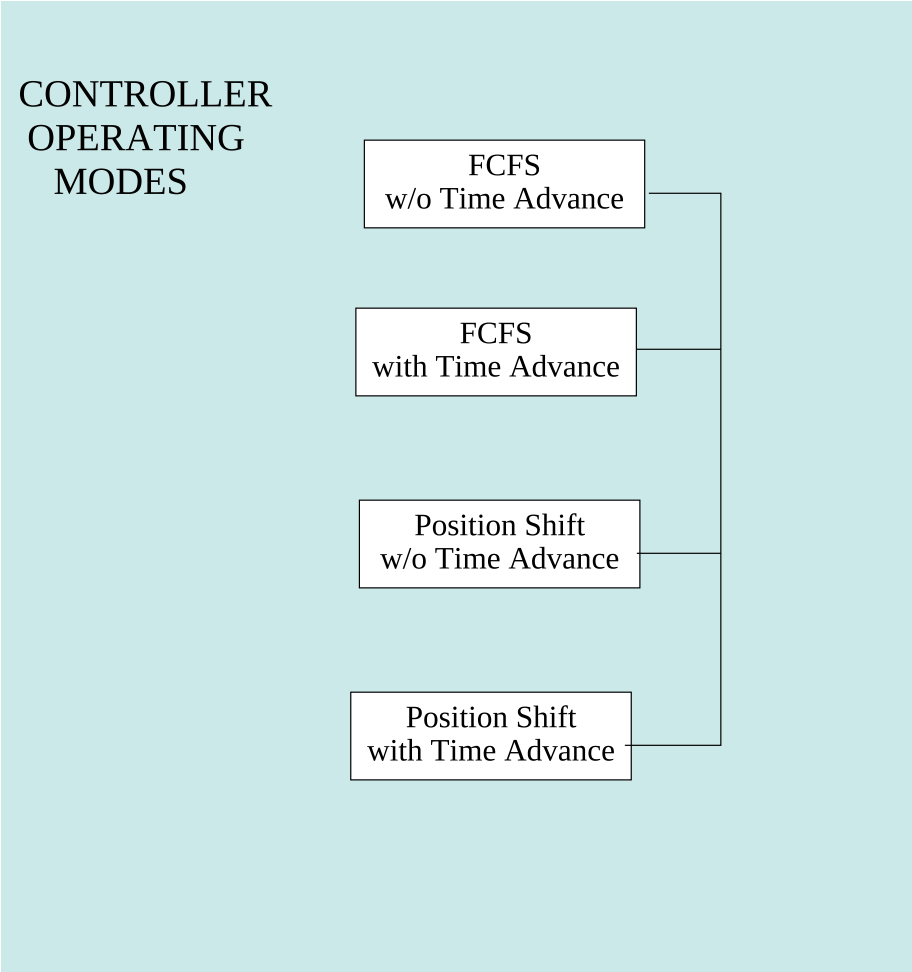


Figure 10: Example of operating modes using the TMA model

model. The TMA can provide scheduling using First-Come-First Served (FCFS) without time advance, FCFS with time advance, and position shift with and without time advance.

A second part of a SpecTRM-RL model is a specification of the component's view of its interface with other components. The supervisory interface consists of the controls by which a supervisor directs the control component and the displays or other means of communication by which the component relays information back to the supervisor. In addition to the supervisory interface, a control component usually has an interface with the controlled system, which includes inputs and outputs from and to sensors and actuators. Note that these interface models are simply the view that the component has of the interfaces—the real interface (such as the ATC controller's planview display) may contain different information due to various types of design flaws or failures. By separating the assumed interface from the real interface, we are able to model and analyze the effects of various types of errors and failures.

The third part of a SpecTRM-RL model is the controller's model of the controlled system. The description of a simple component may include only a few relevant states. If the controlled process or component is complex, the model of the controlled process may itself be represented in terms of its operational modes and the states of its subcomponents. If, during the design process, components that already exist are used, then those plug-in component models would be inserted into the SpecTRM-RL model.

As with all state-machine models, transitions are governed by external events and the current state of the modeled system. In SpecTRM-RL, the conditions under which transitions are taken are specified separately from the graphical depiction of the state machine. We have found that the behavior of real systems is too complex to write on a line between two boxes. Instead, we use a form of logic table we call AND/OR tables. Figure 11 shows an example specification for a transition.

Runway-Not-Assigned $\longrightarrow$ Prefer-1L

Fast-Aircraft-Model.Engine-Type[A] **in state** jet	F	F
Aircraft-near-JEN[A] **in state** yes	T	·
Aircraft-near-UKW[A] **in state** yes	·	T
Fast-Controller-Assigns-Runway[A] **in state** Not-assigned	T	T

Figure 11: An example transition in SpecTRM-RL model

Required controller behavior may be based not only on the most recent measurement of state variables and the assumed current state of the process but also on the previous state and past variable values (both inputs and outputs). We specify these values using the function PREV.

Before we describe our DFW TRACON model, a comment on the amount of work involved in constructing such models might be useful. Some sort of model is necessary to perform hazard analyses, and complex systems will necessarily require complex models to get any real value or important information from the analysis process. Although the amount of work involved in constructing these models is significant, starting from scratch is not always necessary. In the case of analyzing upgrades to the ATC system, most of the models can be reused because most of the components will not change. In addition, the models of the components that will be altered can be designed in such a way that modules or pieces can be substituted for proposed changes while the rest of the model remains the same. To accomplish this goal, some care has to be taken when creating the original model to determine what parts are likely to change and to modularize these parts so that substitutions can easily be made. Our modeling language assists in this process by providing facilities for specifying macros and functions that are "called" from the main model and by the basic separation into components and independent (parallel) state machines.

3.2 The DFW TRACON Model

In this section, we describe the models we built for FAST and relevant parts of the DFW TRACON.

3.2.1 Overview of CTAS

The Center TRACON Automation System (CTAS) provides automation tools—the Traffic Management Advisor (TMA), Descent Advisor (DA), and Final Approach Spacing Tool (FAST)—for planning and controlling arrival air traffic. CTAS generates air traffic advisories designed to increase fuel efficiency, reduce delays, and provide automated assistance to air traffic controllers in achieving acceptable aircraft sequencing and separation as well as improved airport capacity. CTAS must accomplish these goals without decreasing safety or increasing controller workload.

In the U.S. ATC system, the airspace of the 48 contiguous states are divided into twenty areas approximately 400 miles across. These areas, known as Air Route Traffic Control Centers (ARTCCs) or "Centers" for short, are further divided into sectors. The Traffic Management Advisor assists the Traffic Management Coordinator at a Center optimize the arrival traffic flow and create a plan. At the same time, the Descent Advisor assists the air traffic controllers of each sector by probing for and resolving conflicts between aircraft and by providing air traffic control advisories to carry out the Traffic Management Coordinator's plan.

A separate component of CTAS assists controllers handling arrival air traffic in an area within 40 miles of a major airport. Within this TRACON (Terminal Radar Approach Control) area, the Final Approach Spacing Tool (FAST) assists approach controllers to assign aircraft to runways as well as sequence and schedule aircraft onto the final approach to the runway.

CTAS integrates these functions and thereby provides assistance to air traffic coordinators and controllers in both Centers and TRACONs. Moreover, it can incorporate the actions of controllers by refreshing advisories automatically whenever it receives controller inputs or detects unplanned events. To insure accuracy, CTAS makes use of highly sophisticated performance models of the major aircraft types encountered at Centers and TRACONs including jets, turboprops, and piston engine aircraft. Also, each element of CTAS adapts to changes in the traffic situation, air traffic controller imposed constraints, and pilot and airline preferences existing at a particular Center or TRACON by using what is called *adaptation* data. Each tool provides a distinct benefit while the entire suite of tools greatly improves the coordination between sectors and facilities.

3.2.2 What We Modeled

We used our modeling language SpecTRM-RL to model the version of FAST Build 2 for the Dallas/Ft. Worth TRACON. Along with our model of FAST, we needed to produce models of the other components with which FAST interacts, including a model of the Radar Data Processor (RDP), the Flight Data Processor (FDP), and the STARS database, National Weather Service weather data, and the TRACON (human) controllers (see Figure 12). Figure 13 shows how these components fit together and communicate within the ATC system.

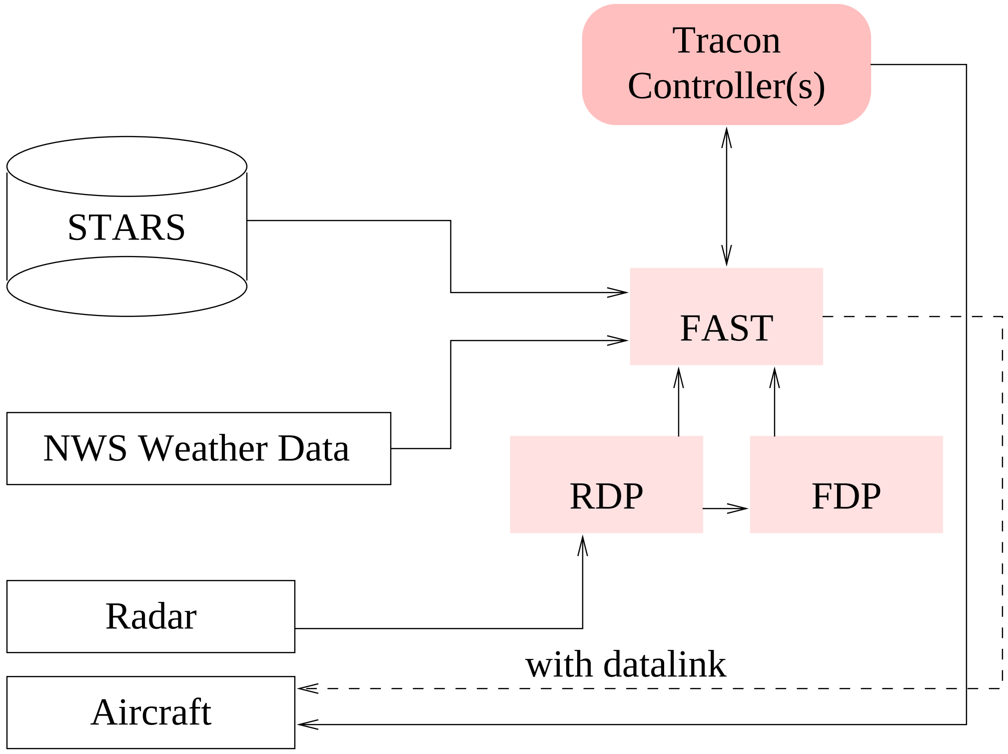


Figure 12: High level view of the aspects of CTAS we modeled.

Because we were limited in our information about DFW and FAST, we built a generic TRACON model similar to DFW that presents the elements that would be present in a real model. In some cases we simplified. For example, we did not have access to the database of aircraft performance characteristics included in FAST, so we created some generic aircraft performance models. In other cases, we simply made up information, such as the adaptation data for FAST executing at DFW.

In our model of DFW, there are four corner posts (Glen Rose (JEN), Bonham (BYP), Bowie (UKW) and Cedar Creek (CQY)), four final approach fixes (Siler, Searc, Deitz, and Delmo), and two runways (1L and 1R). There is one standard approach defined from each

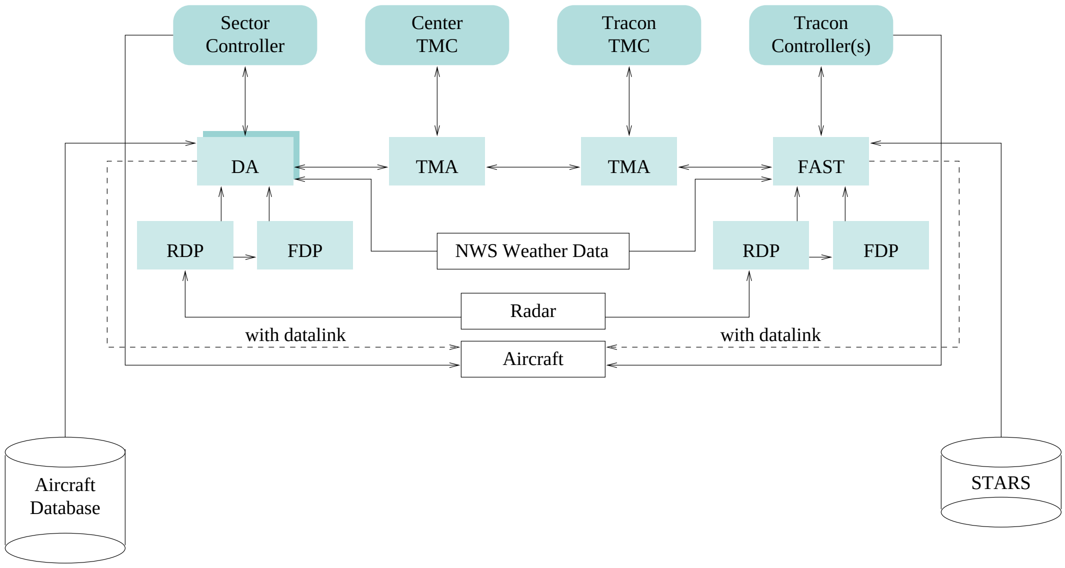


Figure 13: Overview of how the ATC components communicate.

final approach fix to each runway for a total of eight standard approaches in all. Figure 14 shows the DFW standard approaches we used, and Figure 15 shows the innermost area within the Deitz, Delmo, Siler, and Searc fixes.

3.2.3 The FAST Model

Our model of FAST is based on the description of Passive FAST Build 2. As stated earlier, a SpecTRM-RL model includes the component's operating modes (if this abstraction is useful for the component) and a model of the process for which the component generates control commands—in this case the aircraft and airspace.

The FAST operating modes (Figure 16) include the accept mode—whether it is using auto-ready or auto-accept—and, for each corner post, if aircraft gated through that corner post are using the accept-mode policy. We also model the state of various interface modes specific to FAST Build 2, including the overall state of the display and various pop-up windows that cover parts of the PVD.

The Aircraft Model (Figure 17) includes the parts of the state of the aircraft that is needed by FAST to generate advisories. There is one model for each aircraft. These models include the aircraft's size and engine type, runway assignment, controlling sector, and whether the aircraft has been assigned a priority by the controller.

The aircraft model includes ETA and other kinematic data, as well as aircraft performance models that are generated and used for 4D trajectory synthesis. In the interest of time, we implemented the trajectory synthesis algorithms in C and built a table that we used to drive simulations.

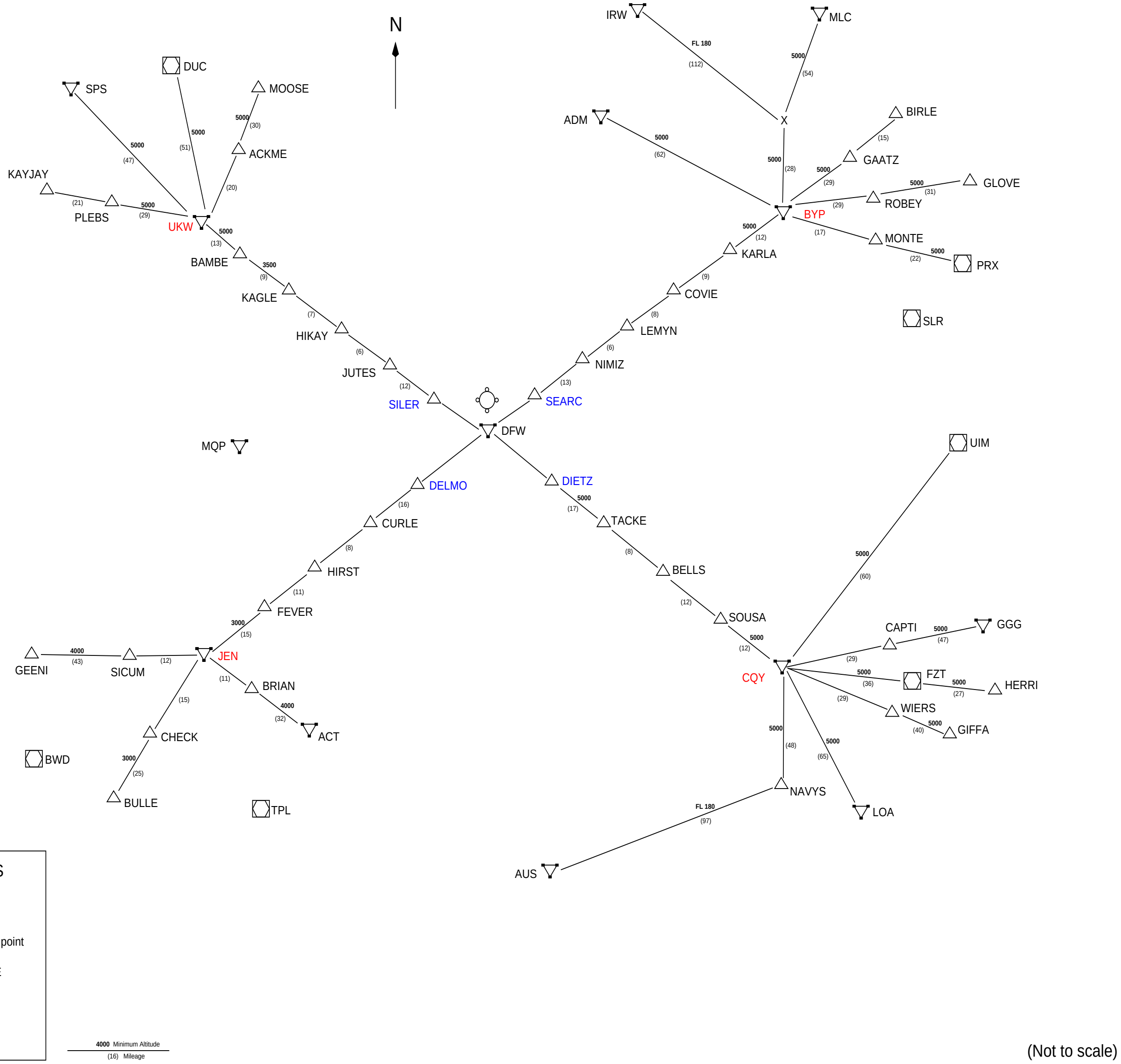


Figure 14: DFW Standard Approaches

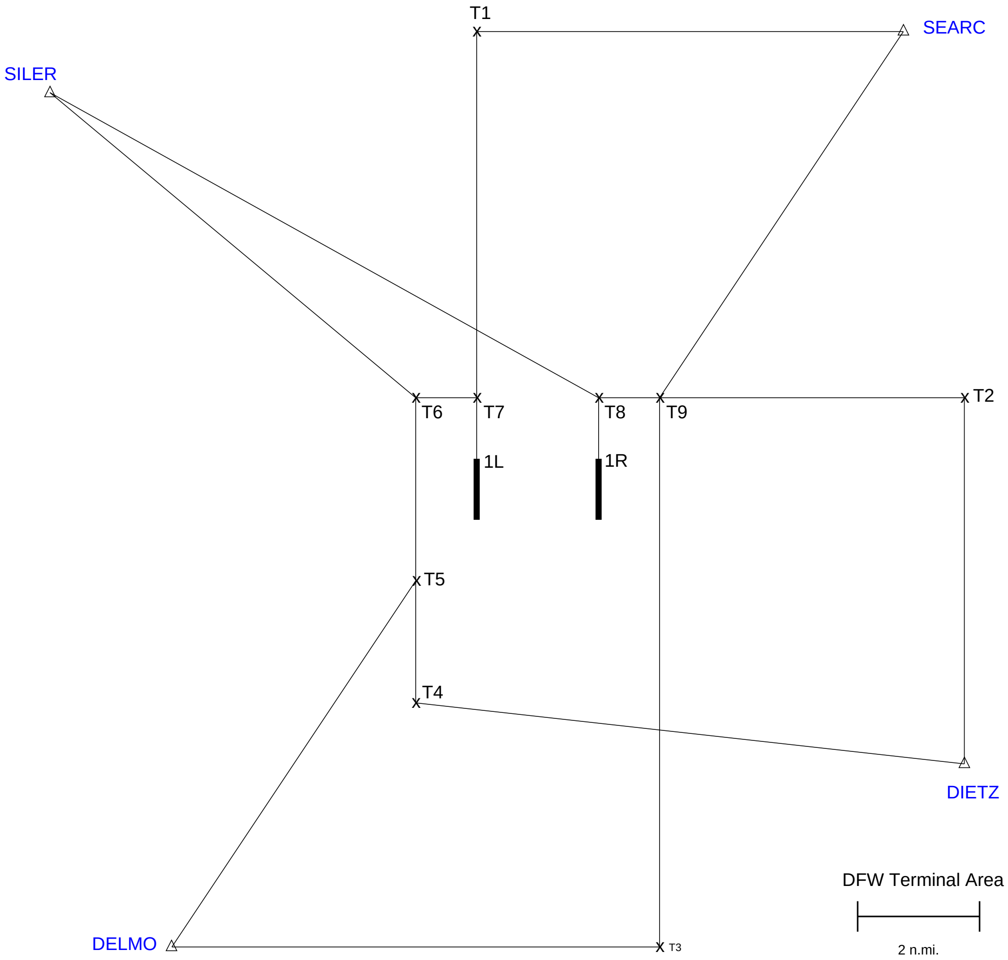


Figure 15: Our model of the standard approaches within the four final approach fixesL Siler, Searc, Delmo, and Deitz

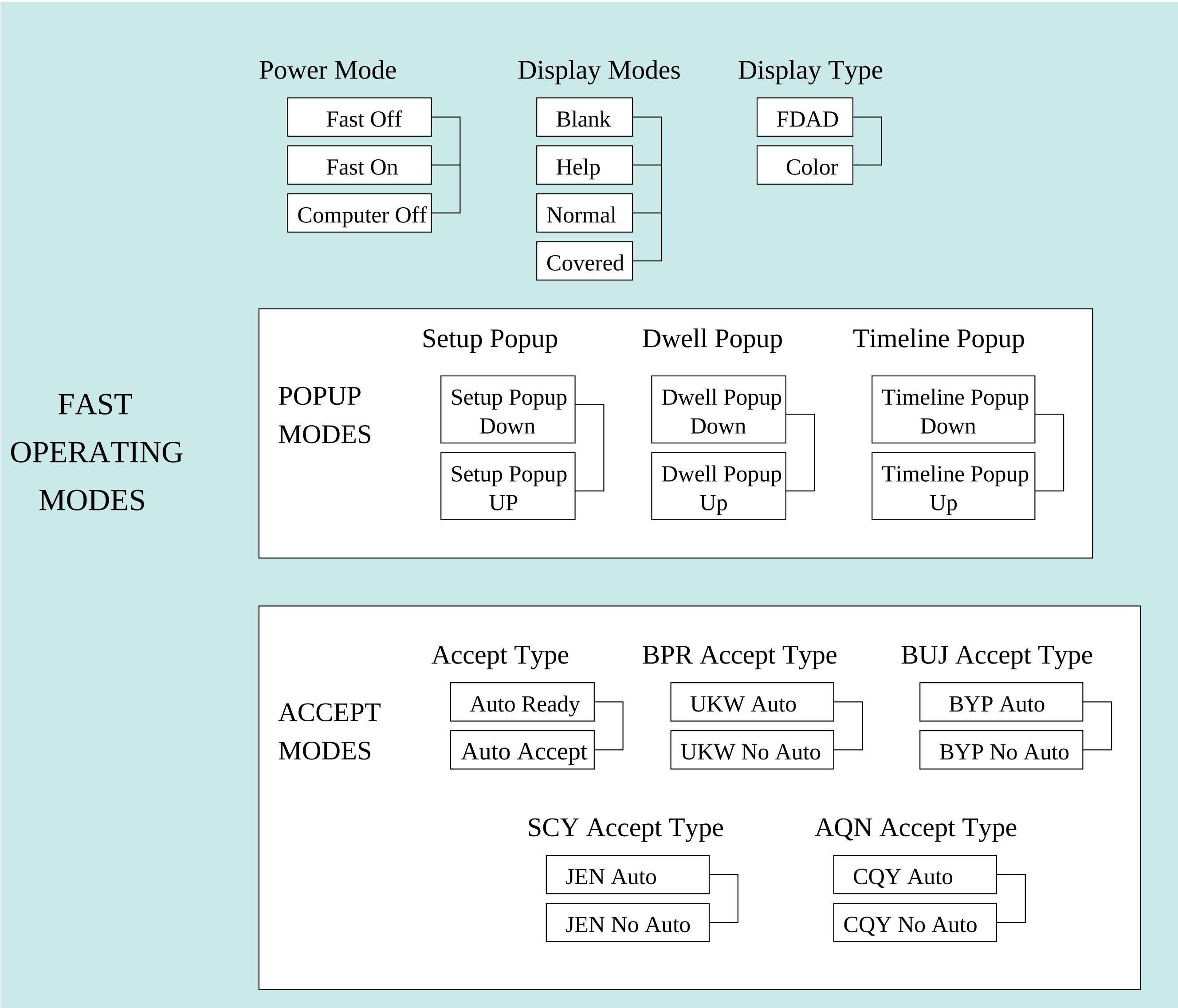


Figure 16: FAST Operating Modes as modeled in SpecTRM-RL

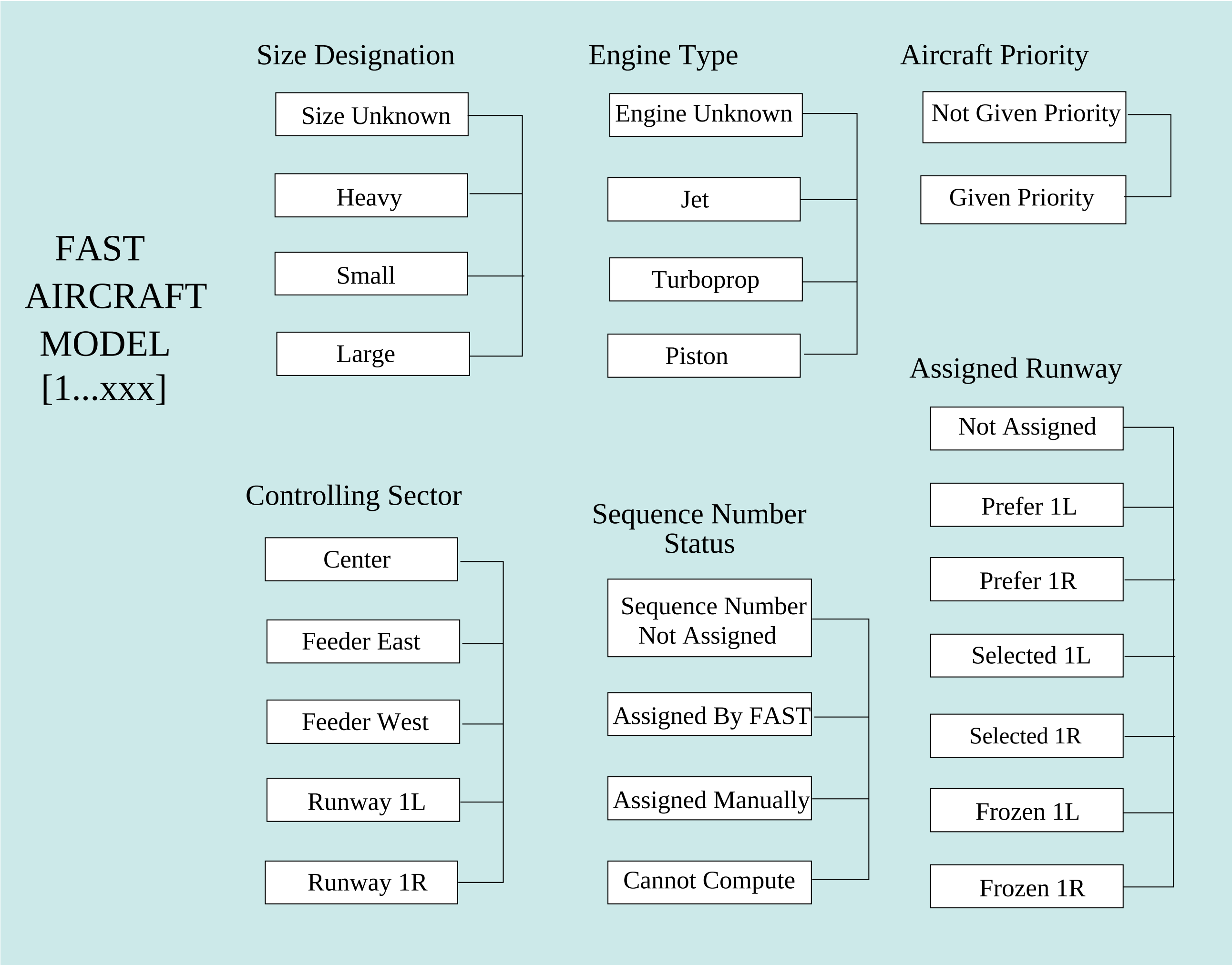


Figure 17: FAST Aircraft Model as modeled in SpecTRM-RL

3.2.4 Models of the Other System Components

Different levels of detail are possible and necessary for the various system component models. Our model of FAST is very detailed while the RDP model requires less information.

The operating mode abstraction is not necessary for our RDP model (Figure 18). RDP processes and smoothes radar data and then outputs the clean radar data to the controllers PVD (or, in this case, FAST). RDP may also generate some alarms: Minimum Safe Altitude Warning, Restricted Airspace Intrusion, Flight Plan Conformance Monitoring, and Potential Conflict.

We folded the STARS database into the FAST model along with the adaptation data. A limited form of FDP information is used by FAST, but this information is incorporated also into the FAST model instead of building a separate model of FDP.

Finally, a simple weather model was incorporated into the 4D path synthesis computation. We only partially modeled the other components of CTAS (DA and TMA). The graphical parts of these models are shown in Appendix E.

3.2.5 Putting the Models Together

SpecTRM-RL models are blackbox and the semantics are defined such that the only way in which components share information is through specification input and output interfaces. There are no broadcast events. This blackbox nature of the models allows us to examine and perform preliminary analysis on single components and partially completed systems, which greatly simplifies the construction of the models and the specification and understanding of the overall system behavior.

3.2.6 Handling Adaptation Data, STARS, etc.

Adaptation data for FAST includes such information as preferred runway assignments, decision criteria for changing runway assignments, freeze horizons, etc. We encapsulated the adaptation data as functions and macros in the SpecTRM-RL model of FAST, which allows:

1. Isolating the adaptation data in one place in the specification so it can be easily changed.

2. Checking the consistency and logical (mathematical) completeness of the adaptation as part of our automated completeness and consistency analysis.

3.2.7 Ease of Incorporating and Analyzing Additional Upgrades and System Changes

The modular nature of SpecTRM-RL models as well as some features of the language, such as the use of macros and functions, assist in modeling and hazard analysis of alternative

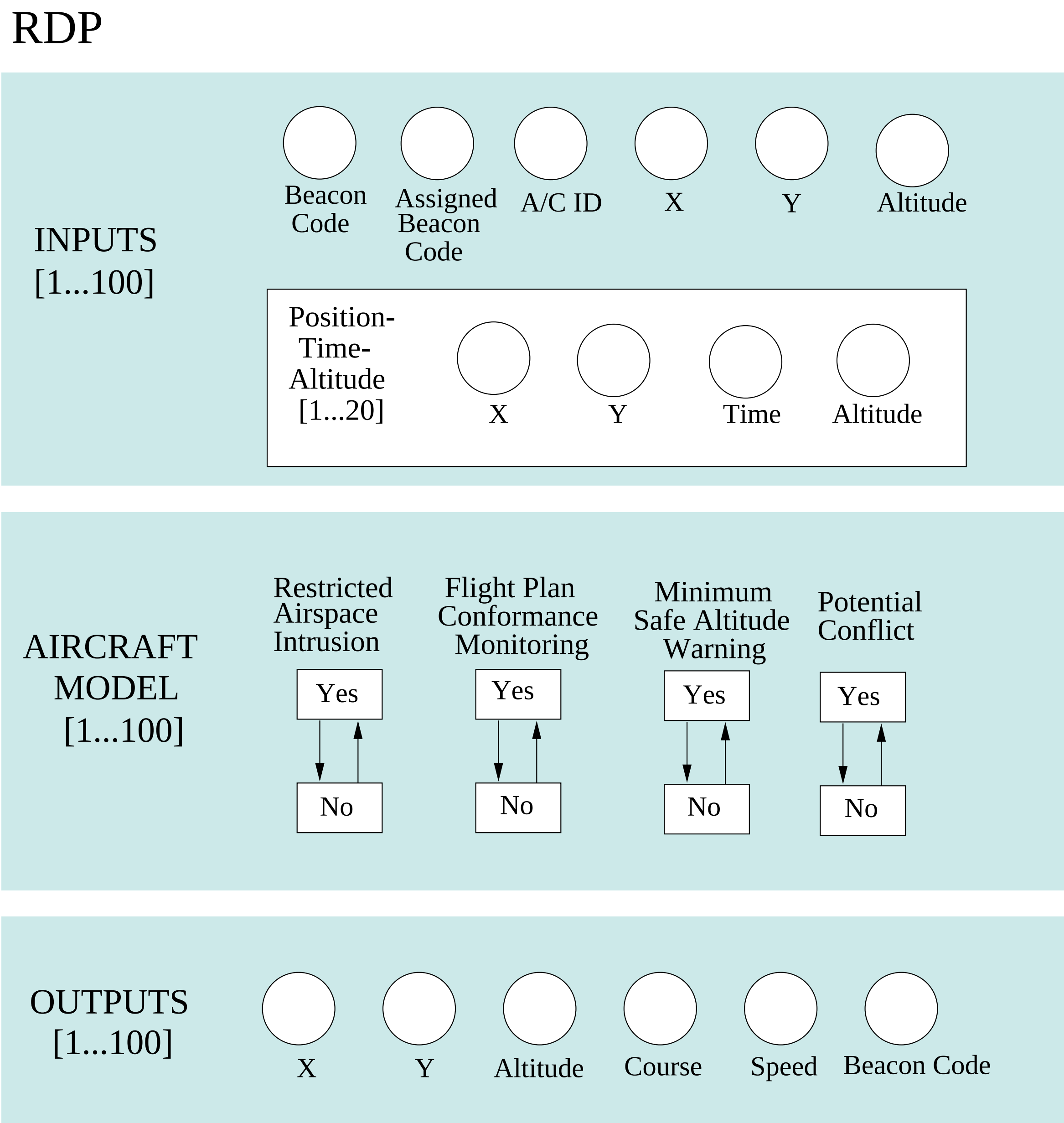


Figure 18: SpecTRM-RL Model of RDP

designs. For example, consider the changes necessary if Automatic Dependent Surveillance (ADS) replaces the FDP. To model and examine the safety-related consequences of such a change, the single RDP component would be replaced with an array of ADS components—one for each aircraft. A first analysis of this change would indicate that a new process is needed to replace the alarms that RDP generates like MSAW, RAI, Flight Plan Conformance, and conflict alerts.

In addition, although the kinematic outputs (s, y, *alt*, *speed*, *heading*) of RDP and ADS are functionally the same, the failure modes of the two systems are quite different. While the probability that a single aircraft will be lost (not reporting) is low in an RDP system, it is higher in an ADS-equipped system. Conversely, a catastrophic failure in RDP might yield no radar information whereas a similar type of failure may not be as likely with ADS. A controller is more likely to detect a loss of all aircraft information than a loss of only one aircraft, particularly if the loss of a single aircraft occurs in certain conditions, such as during an automatic handoff.

4 Controller Task Analysis

The idea of modeling the controller tasks in such a way that they can be simulated along with the model of other system components arose from our work with the TAP (Terminal Area Productivity) Project at NASA Ames. The TAP project is looking at terminal area procedures in order to determine the safety ramifications of using data links in addition to voice contact to communicate trajectories and routing information between the air traffic controller and the aircraft.

CTAS relies on both humans and computers to provide safe and efficient aircraft advisories. When developing a system like CTAS and examining its ramifications on safety, it is important to examine the interaction of the humans and computers. The addition of automation often greatly changes the demands on humans. Through our modeling techniques, we are able to get a better grasp of these changes and can inspect the system to ensure that the changes do not create new avenues for accidents to occur.

4.1 Controller Task Modeling Language

The first step in examining these interactions is to perform a task analysis for both the current system and the proposed upgrades. The task analysis 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.

Figure 19: Key for reading Controller Task Models

Figures 20, 21, and 22 show such a task analysis[3] for the handoff procedure that occurs when one controller passes the control of an aircraft to another controller (see Figure 19 for a key to the notation). The handoff procedure involves communication between the controller currently controlling the aircraft, the next controller to control the aircraft, and the pilot. In order to get a better idea of their interaction in the handoff procedure, we modeled this task from each individual's point of view.

The language for modeling the task is our own and was created for the TAP project. The underlying model is basically a state machine, like the rest of our models, but it is depicted differently because we felt this format was more helpful in understanding the operator tasks. The normative actions are seen on the main horizontal axis of each person's task model. Any non-normative behavior diverges from this main axis until the situation

[3]Color is an important feature of our models and visualizations. The readers of black-and-white versions of this report may have difficulty understanding them.

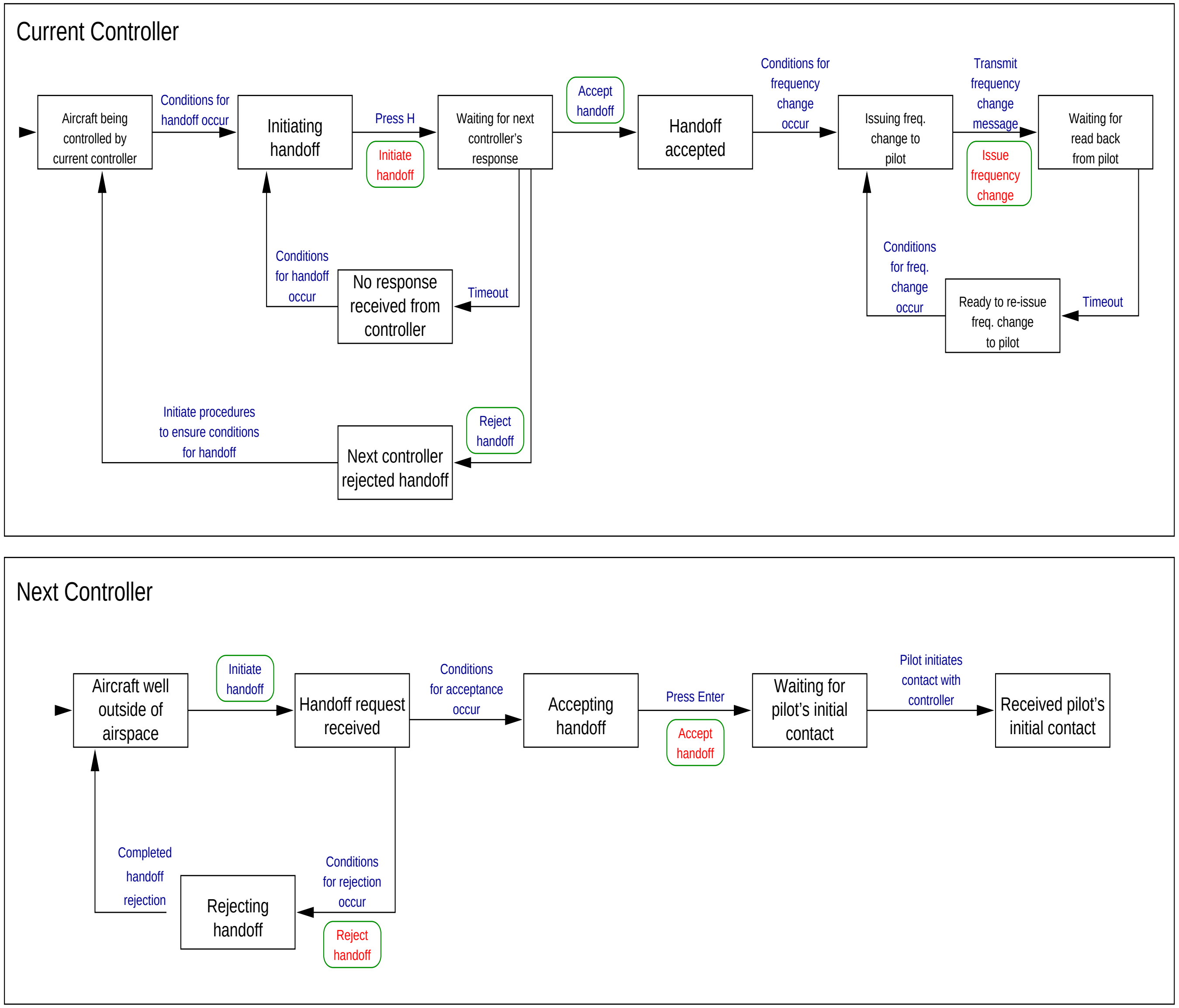


Figure 20: Model of Current Controller and Next Controller

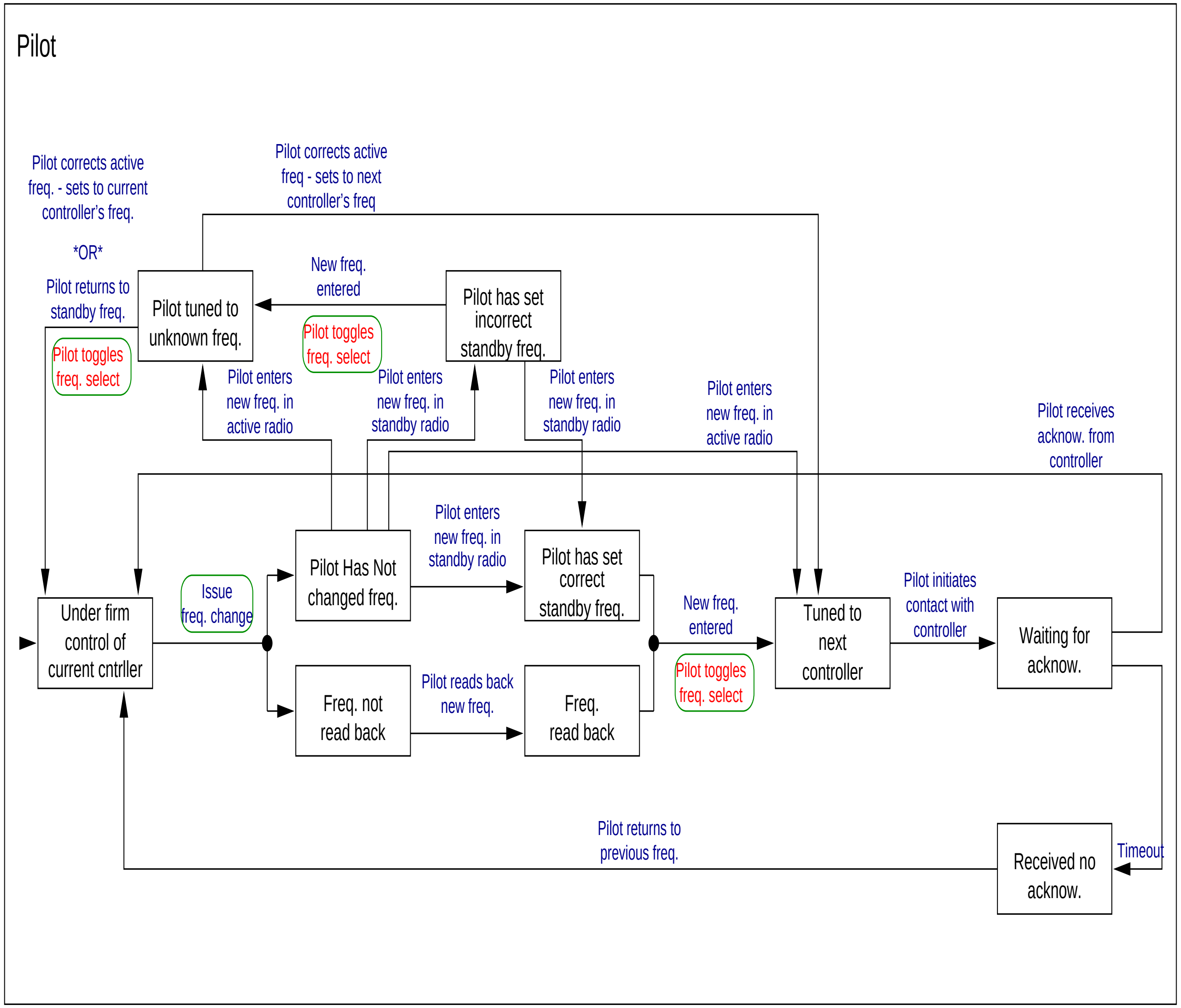
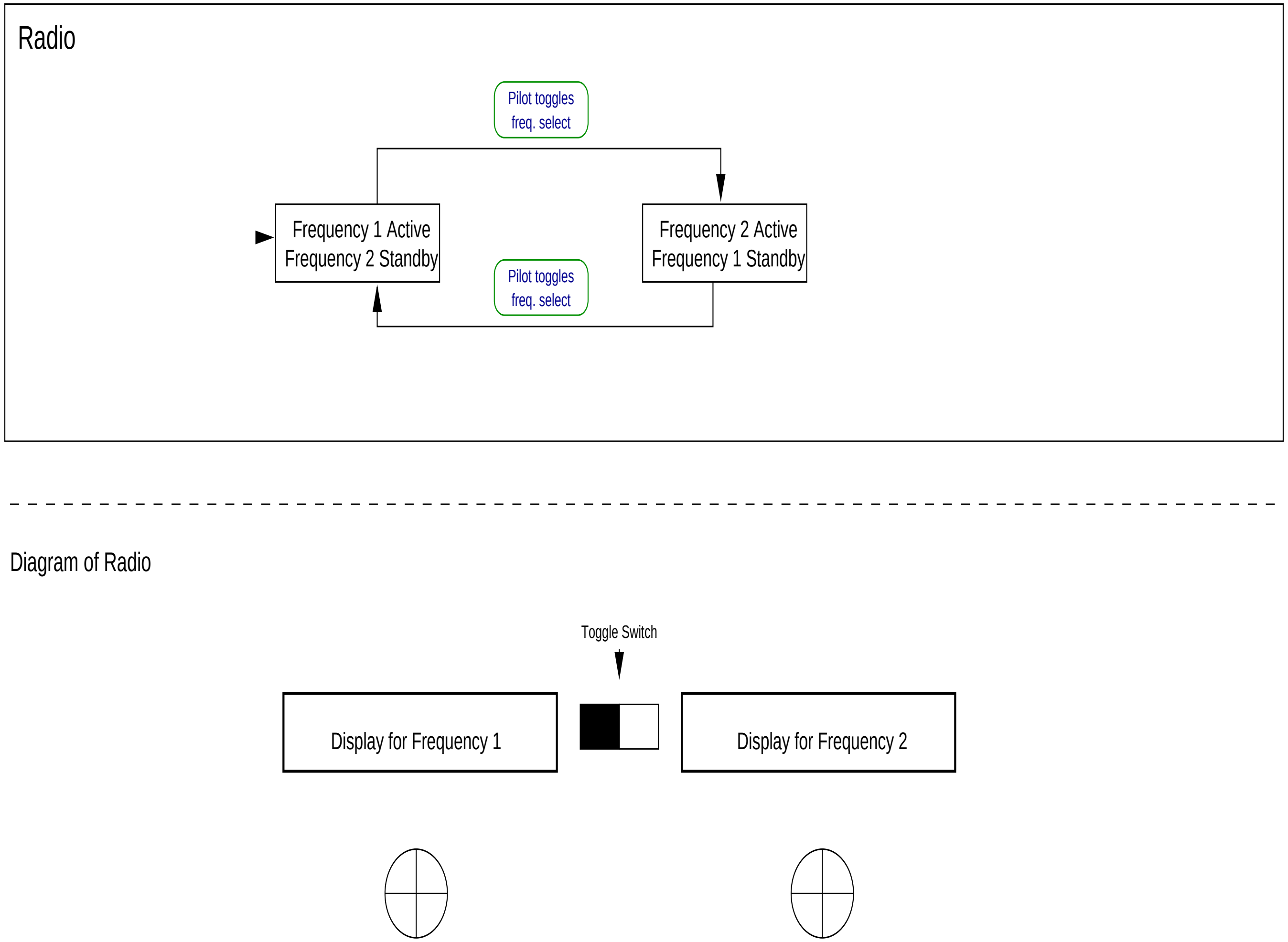


Figure 21: Model of Pilot



Note: The radio had two displays and two dials. One display and dial monitors frequency 1 and the second display and dial monitors frequency 2. There is a toggle switch that allows the pilot to chose between frequency 1 and frequency 2 as the active frequency. The unselected frequency is referred to as the standby frequency.

Figure 22: Model of aircraft radio

has been corrected and the normative procedure resumed. Color is used in the model to differentiate between events and actions. The triggering events for a transition are shown in blue text above the transition while any actions resulting from the transition are shown in red text beneath the transition. A green outline around text denotes that this is a communication point between two components in the model. For example, in Figure 20 there is a green outline around the action *Initiate Handoff* in the Current Controller model and a green outline around the trigger *Initiate Handoff* in the Next Controller model. This part of the model denotes that the action of initiating a handoff in the Current Controller model causes a transition to occur in the Next Controller model.

Another form of our controller task models, shown below, includes cognitive, perceptual, and motor requirements.

4.2 Modeling Tasks Using SpecTRM-RL

Connecting these models to our FAST model, we are able to simulate the interaction between the humans and the computer. The analyst is able to closely inspect the ramifications of certain automation and human task design decisions on overall safety. For example, we know that humans do not make good monitors and that relegating humans to a monitoring role can lead to accidents. If the controllers at any time need to take over full control, we want to ensure they have enough information about the system state to do so effectively. Our models also allow the analyst to examine the cognitive, perceptual, and motor demands on the humans, at least with respect to the normative tasks.

From the task analysis, we created a model of the handoff procedure using SpecTRM-RL. There are aspects of nondeterminism that are inherent in the controller's behavior that do not exist in a software system. To accommodate these unique qualities of modeling controller behavior, we had to expand the semantics of some parts of the SpecTRM-RL modeling language. In our task models, we need to be able to represent inputs that motivate the controller to perform a given action. The exact cause of these actions are not necessarily important for our modeling purposes. For example, although there are some physical constraints that determine the time period in which a handoff is initiated, the exact time that the current controller begins this process is determined by many factors, such as the controller's work load. For our purposes, we needed to be able to communicate to the model that these motivating conditions have occurred so that the model execution can progress.

Our slight changes to the semantics of the SpecTRM-RL modeling language do not violate the underlying mathematical model. We are still able to perform the mathematical analyses, such as consistency and completeness checks, forward and backward simulation, or deviation analysis, on a system model that has been augmented with the normative controller task models. While these analyses give the analyst a clearer picture of the system, there are many interesting questions that these mathematical procedures do not answer, such as:

- Are we modeling the handoff procedures correctly under normal operating conditions? Under expected abnormal operating conditions?

- How does communication flow between the controllers and pilot during the handoff?
- What types of cognitive, perceptual, and motor resources are required of the pilot and controllers?
- How does the cognitive, perceptual, and motor load on the pilot and controllers vary throughout the task?
- When looking at two or more proposed handoff protocols, how do the demands on the pilot and controllers change under the different protocols?

The designer of the automation and the controller procedures can answer some of these questions through executing the augmented SpecTRM-RL models and using visualization (animation) on various scenarios to determine the results. The flexibility of the visualization tool in its interaction with the model provides a powerful aid to help the analyst explore the system design more thoroughly.

The first visualization we created for our handoff procedure shows the overall communication flow between the controller currently controlling an aircraft, the next controller to control the aircraft, and the pilot. This visualization highlights the actions (shown in red) that are taken by each person involved in the handoff procedure. Figures 23 and 24 show a snapshot from this visualization. From it, we can determine if we have modeled the handoff procedure correctly. It also allows us to inspect the communication flow between the controllers and the pilot as the steps for the handoff procedure are simulated to the detail of the necessary computer commands to enable the handoff using FAST.

The second visualization (Figure 25) we created allows us to inspect the cognitive, perceptual and motor load on the pilot during the handoff procedure. Barry Crane, a cognitive psychologist at NASA Ames, has broken down the different actions that are required by the pilot during the handoff procedure into the following categories:

- Cognitive resources
- Perceptual resources (aural or visual)
- Motor resources (hand or voice)

From this visualization, we can compare the effects of multiple suggested handoff procedures on the pilot's cognitive, perceptual and motor load to determine which procedures present safety risks.

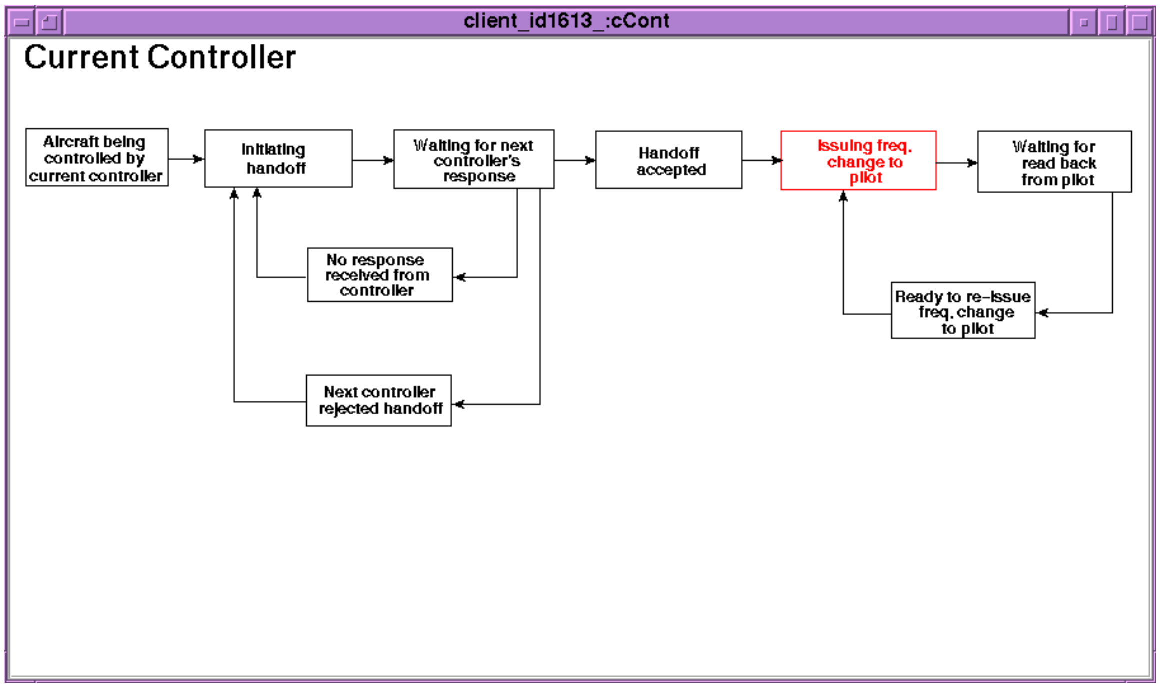


Figure 23: Communication visualization: Current Controller

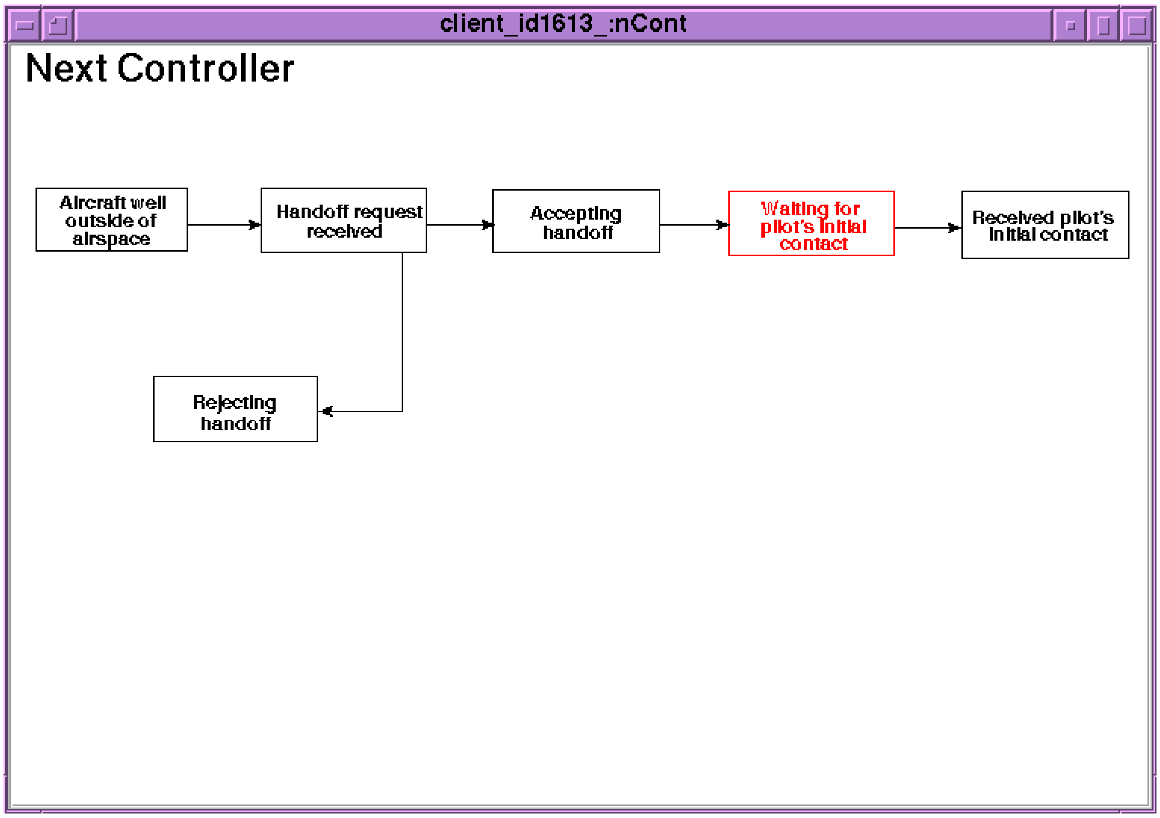


Figure 24: Communication visualization: Next Controller

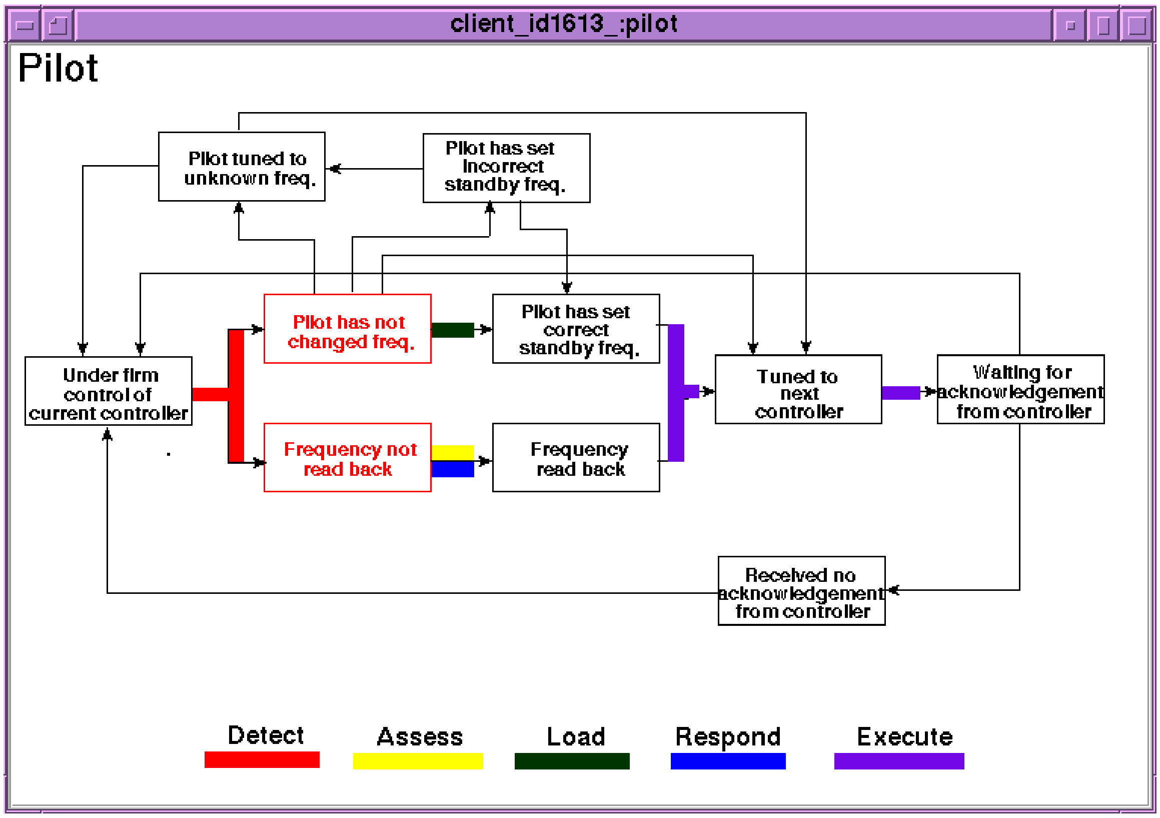


Figure 25: Cognitive and perceptual load on the Pilot

5 Completeness and Consistency Analysis

After the preliminary hazard analysis has been performed and information is available about system hazards, the next step of the process is to document the identified behavioral requirements and constraints and to show that the requirements specification satisfies them. This step also includes demonstrating the completeness of the requirements specification with respect to general system safety properties. Once a blackbox model of the required system behavior has been built, this model can be evaluated as to whether it satisfies design criteria that are known to minimize errors and accidents.

5.1 General Completeness Criteria

The goal of requirements completeness analysis is to ensure that the model of the process used by the software is sufficiently complete so that no hazardous process states are included. Accidents involving computers are usually the result of incompleteness or other errors in the software requirements, not coding errors [Lut92, Lev95]. Jaffe and colleagues [Jaf88, JLHM91, Lev95] have defined a set of formal criteria to identify missing, incorrect, and ambiguous requirements for process-control systems. Some of these criteria must be satisfied by all such systems; they arise from the basic properties inherent in a process-control system. Other criteria involve heuristics for finding flaws that frequently lead to accidents and can be used to improve the specification by examining, within the context of the particular physical process being controlled, properties that are often present in such systems. Some of the criteria are related to human-computer interaction such as providing appropriate feedback during graceful degradation and completely specifying preemption logic when multi-step operator inputs can be interrupted before they are complete. Additional constraints related to human-computer interaction are described in Section 9.

In order to make the general semantic analysis procedures applicable to any blackbox, behavioral requirements specification, we devised a formal modeling language (RSM) independent of any specific, existing requirements language that is an abstraction of most state-based specification languages. The criteria are defined in terms of requirements for completely specifying the parts of this abstract state machine [JLHM91].

A few of the Jaffe criteria were derived from mathematical completeness aspects of the formal RSM model underlying SpecTRM-RL, but most resulted from the experience Jaffe had in building such systems over a large number of years. These lessons learned were used to define design criteria for an RSM model.

Robyn Lutz [Lut93] at NASA JPL applied the criteria experimentally in checklist form to 192 safety-critical errors in the Voyager and Galileo spacecraft software that had been identified as safety-critical. These errors had not been discovered until late integration and system test, and therefore they had escaped the usual requirements verification and software testing process. The criteria identified 142 of the errors. The errors not identified involved design and thus were not the focus of the criteria and the checklist. Any such after-the-fact experiment is always suspect, of course; no proof is offered that these errors would have been found if the criteria had been applied to the requirements originally. But the fact that they were related to so many real, safety-critical errors is encouraging. It is not necessarily surprising, however, since most of the criteria were developed using experience with critical errors, incidents, and accidents in real systems.

The Jaffe criteria have been made into checklists by government and industry and are being used in a wide variety of applications such as radar systems, the Japanese module of the Space Station, and review criteria for FDA medical device inspectors.

A complete list of the criteria are included in Appendix D. Briefly and informally, they involve:

- **State Completeness**: Checking for completeness of transitions and default values during normal and non-normal operation, including startup and shutdown.
- **Input and Output Variable Completeness**: Checking that all information from the sensors is used by the controller and that legal output values never produced by the computer have not been inadvertently omitted.
- **Human–Computer Interface Completeness**: Checking criteria specifically related to the interaction between the computer and operator.
- **Trigger-Event Completeness**: Checking the conditions specified for the values and timing of inputs from the sensors that trigger a state change in the controller's model of the process. Criteria here ensure the complete description and handling of all inputs including essential value and timing assumptions about these inputs. The software must be prepared to respond in real time to all possible inputs and input sequences—there must be no observable events that leave the program's behavior indeterminate—and to respond to input capacity and overload conditions.
- **Output Specification Completeness**: Checking that the conditions specified for the outputs of the software are complete with respect to timing and value including environmental capacity, data age, and latency requirements.
- **Output-to-Trigger-Event Relationships**: Checking the complete specification of the relationship between inputs and outputs including feedback loops and graceful degradation.
- **Transition Completeness**: Checking properties of the paths between states including basic reachability, recurrent or cyclic behavior, reversible behavior, reachability of safe states, preemption of transactions, path robustness, and consistency with required system-level constraints.

Jaffe originally identified 26 completeness criteria, which we have now extended to over 50. Most of these criteria can be checked by inspection. In order to assist in this inspection and to assist the specifier in writing complete specifications from the beginning, we have designed SpecTRM-RL to enforce (where possible) the criteria in the language syntax and to make others more easily checkable in the resulting models. For example, SpecTRM-RL requires that all components of the controlled system model have an UNKNOWN state, which is the default for startup and after transitions from normal processing to any type of temporary or partial shutdown. The controlled system can usually continue to change state when the computer is shut down, and the software model of the process must be updated at startup or restart to reflect the actual process state. Many accidents have occurred in systems where the software assumed the status of the process had not changed since the computer was last operational and issued commands based on this erroneous information. In translating from our RSML specification of TCAS to a SpecTRM-RL model, for example, we found such an omission. When TCAS is turned on, the altitude layer is initialized as below 400 feet (where oral alert annunciations are inhibited). Although TCAS is normally turned on while the aircraft is on the ground and the initialization is thus correct, it is possible to turn it on while at a higher altitude layer, and there are unusual scenarios where this event might occur.

The checking of a few of the criteria need to be aided by automated tools for large and complex models. Our Consistency and Completeness Analysis tool [HL96] checks two criteria that are difficult to check manually: (1) consistency (no two transitions out of a state are satisfied simultaneously, that is, the behavior described is deterministic) and (2) transition or mathematical completeness (there is a behavior defined for every possible input and input sequence). We used this tool while constructing the CTAS models and found that even when the models were very incomplete, it yielded useful information about the transitions that had been specified. In addition, as stated earlier in this report, the tool allows checking the adaptation data for completeness and consistency.

Using the automated completeness checking tool on our models, we found 48 cases where transitions out of a state were incompletely specified. For example, we had a state AUTO-READY indicating that FAST would query the operator to accept aircraft that fly over cornerposts. We had not specified how FAST would react to a command to go into AUTO-ACCEPT (in which FAST automatically accepts aircraft without querying the controller) under the following situations:

Power-Mode **in state** Fast-On	T	T	F
Setup-Popup **in state** Setup-Popup-Up	T	F	·
Display-Type **in state** Color	F	·	·

in other words, if the display type is not color, the setup menu is not being shown, or the power is off. Although the last case is obviously not realistic (the controller cannot select anything if the power is off), the other two cases are reasonable. Spurious inputs, keyboard shortcuts (when the menu is not being displayed), latent inputs, and so on need to be handled.

We also checked for nondeterminism in our model but did not find any instances of it.

5.2 Timing Constraints

The timing analysis we were able to do was limited by the amount of information we had about CTAS. But we did take the system requirements specification (*Center TRACON Automation System, Build 2, System Specification*, Revised Draft, 14 February 1997, Corrected Copy, MIT Lincoln Laboratory) and examine it for completeness with respect to the specification of the timing requirements.

The requirements that we examined are given on pages 17-21 of the document for all functions and central monitoring and control (M&C), on pages 67-68 for FAST processing, and on pages 93-94 for the TMC-GUI (Traffic Management Coordinator Graphical User Interface).

The questions that we asked relate to the clarity and precision of the specifications, the interactions among the constraints, and the robustness of the requirements. Depending on the interpretation, there would clearly be some problems in building software that actually meets these specifications. These problems could affect the safety of CTAS.

In general, the timing requirements are very incomplete with respect to safety issues (as is the entire document). For example, on page 93, a basic timing requirement is specified for FAST that it shall display requested data to the TMC (response) as a result of the TCA data requests (stimuli) with a mean response time of 3 seconds or less, a 99 percentile response time of 5 seconds or less, and a maximum response time of 10 seconds or less. Where did these numbers come from? Are they safety-related or are they simply for convenience? Can they be changed if the system implementation cannot achieve them or if the underlying technology changes? Are there different times for safety-related responses, i.e., are all responses equally important and needed in the same time frame? If there is a need for "triage," i.e., not all the responses can be accomplished in the average or even the maximum time, what information is the most important to provide, i.e., what are the fail safe requirements for timing? What happens if the deadlines cannot be met?

5.2.1 General Function and M&C Requirements

All CTAS processes or functions, except the M&C functions, must handle at least five different kinds of timing constraints, expressed as "SHALL"s in the document. These constraints are given as "adaptable" deadlines, which we interpret to mean deadlines that depend upon the particular instance (air traffic control site) of CTAS and the particular function. Exceeding a deadline indicates either a function failure or an exceptional situation.

The deadlines are described in the following list. The notation t_i is our own and is used later in this section.

- **Initialization time** (t_{init}): The deadline for completing the initialization of a function.
- **Shutdown time** (t_{shut}): The deadline to shutdown a process after receiving a shutdown command ("Off-line Time").

- **State saving interval** (t_{save}): The earliest time preceding a failure at which it must be possible to restore state.

- **Normal response time** (t_{status}): The deadline for responding to an M&C status query ("Stable Time").

- **Abnormal response time** (t_{delay}): The deadline following "Stable Time" for a response to an M&C status query ("Delayed Time").

Figure 26 shows a state-machine description of a generic essential restartable process, focusing on states and transitions related to the above timing deadlines. The timing-related transitions (timeouts) are shown in blue. This description was inferred from the document, making reasonable educated guesses in several places, as discussed below.

The text labeled $timeout[t_x]$ on a transition specifies the maximum time (t_x) for residing in the state; for example, t_{status} is the maximum time that a process can reside in the *Status-Respond* state. If that time is exceeded, a *timeout* transition occurs to a new state—in the example, to the state Status-Response-Delayed.

Following are some of the questions and issues arising from an examination of the specified timing requirements:

1. Two of the deadlines, t_{status} and t_{delay}, and obviously system safety, are concerned with the *response* to an M&C status *query*. However, the document seems to require that CTAS processes send status data on their own, rather than in response to explicit queries. For example, on page 102, M&C outputs do not include a status query command and, as specified on page 67, the FAST function sends status data periodically.

2. Assuming that M&C does send status queries, can there be more than one query outstanding at the same time? Figure 26 allows only one query at a time. If several are allowed, the timing and failure checking could be very complex, potentially overloading the system and causing a failure.

3. Does the "Delayed Time" deadline include the "Stable Time" already passed?

4. Is a shutdown instruction identical to (synonymous with) pushing the CTAS Stop button?

5. The requirement to restore state to a value at an adaptable time prior to a failure (the reason for the timeout and state saving after t_{save} in the states Normal-Processing, Status Respond, and Delayed-Status-Respond) could be interpreted in a non-tolerant manner as maintaining a state value t_{save} units before a failure *detection* rather than the failure itself. For example, in the figure, a process could be failed in either the *Status-Respond* or *Delayed-Status-Respond* states, and the failure may not be detected until the *Recover-From-Failure* state is entered through a timeout; however, the most recent saved state operation could have been triggered after the failure had

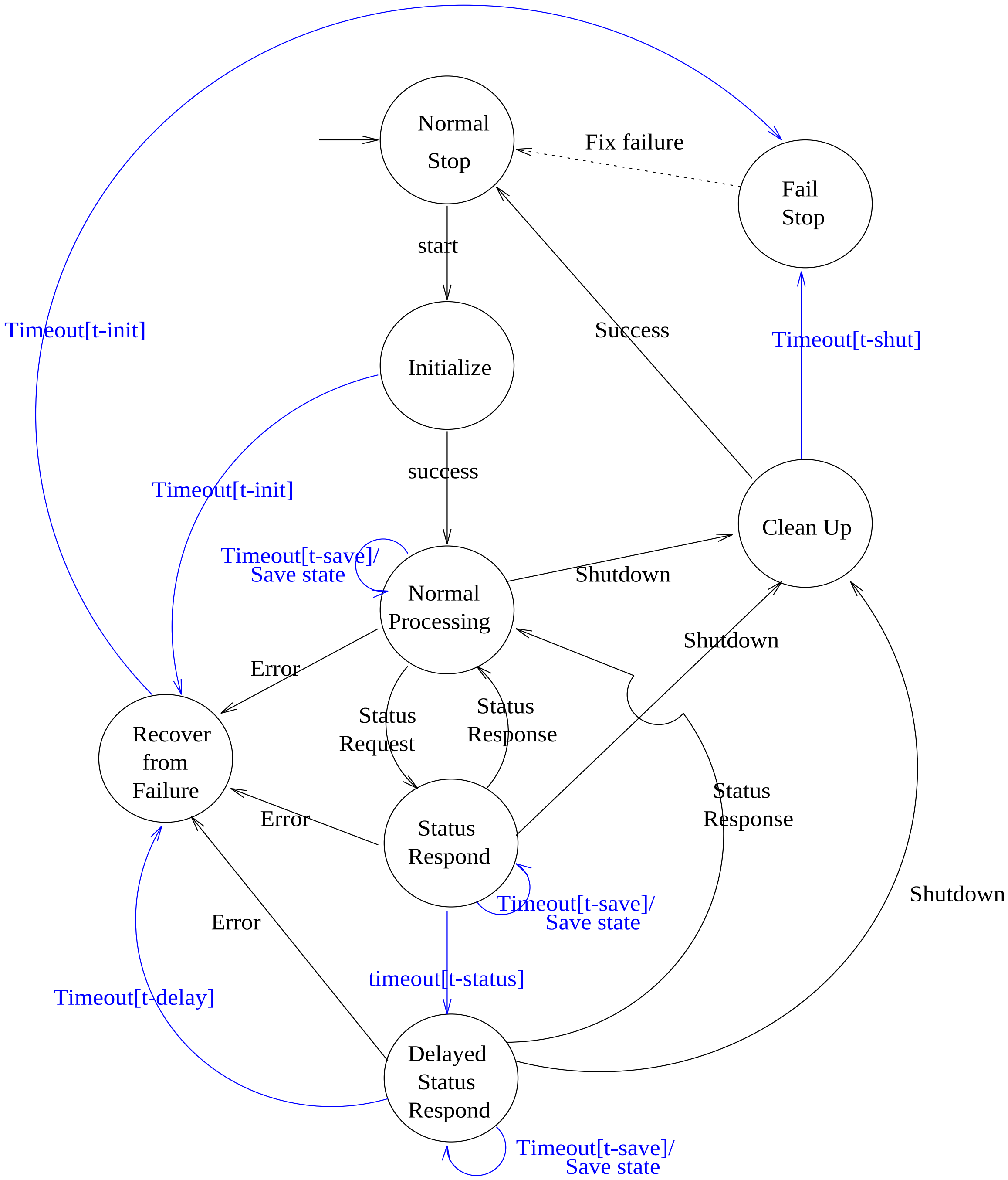


Figure 26: State-machine model description of a generic essential restartable process

occurred. (Of course, keeping a history of saved states would allow access to a recent "correct" state prior to the actual failure.)

6. The dotted transition from *Fail-Stop* to *Normal-Stop* indicates that some other means for recovering from failure are necessary. There is no specification of what those means might be.

5.2.2 FAST Functions

In FAST, the same timing constraint is defined for input, output, and aircraft processing functions: They are to be executed periodically with a period and deadline of six seconds. These functions include:

F1: Calculate a new aircraft arrival plan.
F2: Update arrival plan if a TMC command is received.
F3: Output to TMC-GUI and controller screens.
F4: Send status to M&C.
F5: Update the number of TRACON-visible aircraft based on ARTS radar.

Some of our questions and issues follow:

1. Is function F2 part of F1, or is it an entirely separate update?

2. What happens if the six second deadline is missed? Is this a FAST or CTAS failure?

3. The six second period seems to be "hard wired." What happens if technology changes to make this period shorter or even longer?

4. How does FAST's periodic constraint interact with the global constraints described in Section 5.2.1 above? For example, if the adaptation times t_{status} and t_{delay} are less than six seconds, the system may signal timeout failures even though FAST is working correctly. It might also mean that M&C status queries cannot be issued more frequently than once every six seconds.

5. It is not clear from the document whether the intent is that F1 through F6 are to be performed once every six seconds as a unit or, alternatively, each of F1 through F6 may have its own six second execution cycle. The difference is that the latter interpretation, while perhaps more complex to implement, is more flexible and robust—periods can be changed, functions can be allocated to different hardware processors, and errors may be easier to detect and handle.

5.2.3 TMC-GUI

Some very specific and precise deterministic and stochastic timing constraints are defined for response and feedback time for the GUI. We have not examined these constraints in any detail, except to note that no mention is made of any requirements for checking these constraints or for handling constraint violations during system execution.

6 Simulation and Animation

Visualizations are pictorial representations of information that help convey meaning and explain concepts. The arrangement and portrayal of information can facilitate or distract from learning and understanding.

Information visualization for requirement and systems analysis is concerned primarily with providing different views of a specification so the designer or analyst can understand the system and the system model. They can help explain the relationship and interactions between system components. Because a system specification is the link between the operators, designers, and developers of a system, the information needs to be presented in the most natural manner for the intended user while still maintaining the ability to be easily verified. Visualizations, when effective, reduce cognitive load by highlighting the relevant interactions and aspects of a system.

Most requirements or system specification languages are either completely textual or provide only a single system view. Information is often grouped to make automated analysis easier. Automated tools can check static attributes such as completeness and consistency, but expert review of these specifications is often difficult. Very formal, mathematical languages are often far from the model the expert has of the problem domain and finding errors through human review can be difficult. Useful visualizations provide alternative system views that can help experts verify system behavior with minimal overhead. In addition, when checking a system specification, experts are verifying behavior of many different aspects of a system. Thus, the information they need will depend on the question being asked.

Experience gained from the specification of other systems helped us to build a list of questions commonly asked during the system specification review process. This list includes questions such as:

1. What caused the system to get into this state?
2. What is the relationship between these n states?
3. What states can the system transition to from this state?
4. Is this what I really want the system to do?
5. What happens if the system receives input x?
6. Under what conditions does the system output x?

The requirements specification of the FAST system that we created includes visualizations to help analysts answer these questions. We also want to use visualization techniques in the representation of the output of our automated analysis tools although these interfaces are still under development.

6.1 Visualizations and SpecTRM-RL

Our IB Toolkit is an interface and visualization builder that allows users to build graphical user interfaces (GUIs) and visualizations of SpecTRM-RL models quickly and easily. Very little previous knowledge about interface programming or graphics is required. Two important assumptions made in the design of the IB Toolkit is that SpecTRM-RL users have a basic understanding of the event programming model and that they are familiar with the Tcl/Tk environment.

The IB Toolkit provides a drag-and-drop mechanism for interface and visualization building. There is a toolbox from which users can click to choose objects to draw on an interface canvas. Once an object is drawn on the canvas, an attribute window for that object is popped up on the screen so the user can modify the relevant attributes of the object and attach code that controls the behavior of the object. A visualization is created by drawing and modifying multiple objects on the canvas.

Once the "look" of the visualization is completed, it can be attached to a SpecTRM-RL specification to control its execution, display the execution outputs, or display internal states or actions of the model during execution. Attaching (linking) a visualization to a SpecTRM-RL model using the IB Toolkit is automatic. The toolkit uses a set of subroutine libraries to access the named variables in the model and then creates a set of bindings between the model variables and automatically generated visualization variables. The visualization attributes can be changed based on the SpecTRM-RL model state.

6.2 Pseudo PVD

The first visualization that we created was a simplified planview display (PVD) to help system designers determine the correctness of the 4D trajectory synthesis algorithms, to monitor the behavior of the aircraft in a simulation, and to control the execution of the model (see Figure 27).

The pseudo PVD has three windows, each presenting a different aspect of the 4D trajectory algorithms. The window to the left is a timeline that shows each aircraft's estimated time to landing. The window to the right is an altitude indicator for the planes, which is divided into two sections. The upper, black section is a coarse-grained altitude indicator for aircraft flying higher than 9000 feet. The larger, blue portion is a more finely grained altitude scale. Because we are modeling the TRACON area, we are mainly concerned with the altitude changes of aircraft below 9000 feet. We included the capability of tracking aircraft at higher, cruising altitudes, however, so the visualization can be used for a broader range of analyses.

The main, center window mimics a simplified PVD. The simplified PVD shows only controlled aircraft, the two runways in the model, and the DFW corner posts. We felt the simplified view would be more useful to the system designer than a real PVD visualization—our goal here is to assist the system designer and not to create or evaluate the interface itself. The PVD visualization receives 4D trajectories for each of the controlled aircraft and displays each aircraft and its predicted flightpath. As the model simulation continues, the

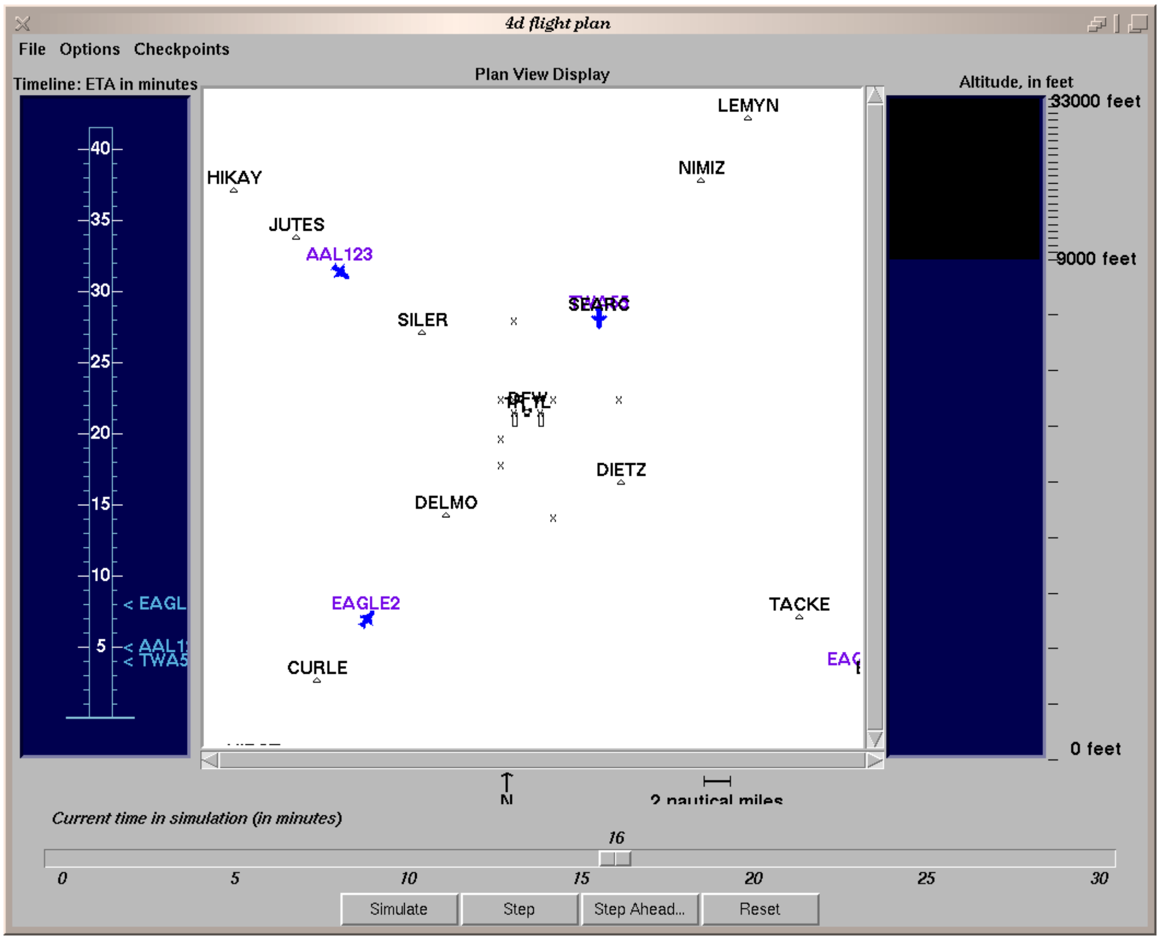


Figure 27: Simplified PVD: The PVD visualization has three windows to help analysts better understand the execution of the specification. The window on the left displays the aircraft time-to-arrival information. The one on the right displays each aircraft's altitude, and the center display shows aircraft flightpaths and horizontal spatial information for each of the planes in the system.

PVD shows the aircraft moving along their projected flightpaths, updating the flightpaths as directed by the model. In addition to the separate altitude indicator in the left window, the pseudo PVD includes grey shadows of the planes to indicate the relative altitudes of each of the planes. The shadowing provides a more complete view of the simulation results in one picture without requiring a context switch to another window.

Because we are concerned with providing visualizations that help answer the questions asked by system designers and reviewers, the pseudo PVD allows the designer to alter the displayed information dynamically during execution. Users have the following options:

- Turn off the 4D trajectory path trace.
- Show the entire path trace from simulation beginning to landing.
- Show just the path trace of where the plane has been.
- Show the last few steps and the next few steps in the path.
- Zoom in or zoom out to any distance from the runways.
- Show labeled waypoints or not.
- Display selected aircraft information such as speed, assigned runway, and sequence number.

The PVD visualization is attached to the SpecTRM-RL FAST system model and controls the simulation of the model. It does this by sending messages to the model that a controller would normally issue during the operation of FAST. The visualization also receives messages from the FAST model so it can update the display as the model executes. FAST decides which runway to assign to each aircraft, and it informs the display about the path each aircraft is predicted to fly.

6.3 State Machine Model

We use a visualization of the underlying SpecTRM-RL model in our simulation and analysis. Each portion of the model is represented in a separate window. For a simulation of three aircraft, we would have four visualization windows: one for each of the aircraft models (Figure 28) and one to display the FAST operating modes. The state machine view of the system model facilitates answering question number three from above. When coupled with the model simulation it helps answer question number five.

What we want to focus on here is the layout and the visualization of the underlying state-machine model. We use principles described by Tufte [Tuf90] in our graphical layout of the state machines. For example, one of these principles is to eliminate unnecessary "grouping" boxes. In Figure 28 the states *Size Unknown, Heavy, Small,* and *Large* are not grouped within a box, but rather are simply denoted as related atomic states by the *Size Designation* label located above the states and the transition arrows between the states. This type of layout reduces the amount of clutter in the display and does not draw attention to or create the artificial visual space that would be present had we put a box around state machines. If there are multiple hierarchical levels and delineating boxes are necessary, we have reduced the contrast between the containing states by bordering these states in light grey.

When the state machine visualizations are coupled with a model execution, an analyst can view the behavior of the model. In the state machine representation, color is used only to denote state activity. The outline and name of a state are drawn in red when, during execution, the model is "in" a state; in black when a state is inactive; and in a muted purple when the status of a state is unknown.

6.4 Transition Tables

While the state machine representation presented above helps analysts answer many questions, we found that the underlying transition tables that govern the behavior of our models are good representations to help answer the question: What caused the system to get into this state? We have included in our simulation the ability to access these tables and to have them dynamically represent the activity of the transitions (Figure 29). During a simulation of the FAST model, single clicking on a state will bring up a window with all the transition tables that influence how the model might transition into that state (see Figure 30). If the state being examined is currently active, the transition that caused the model to change to that state as well as the specific column that enabled the transition are both highlighted.

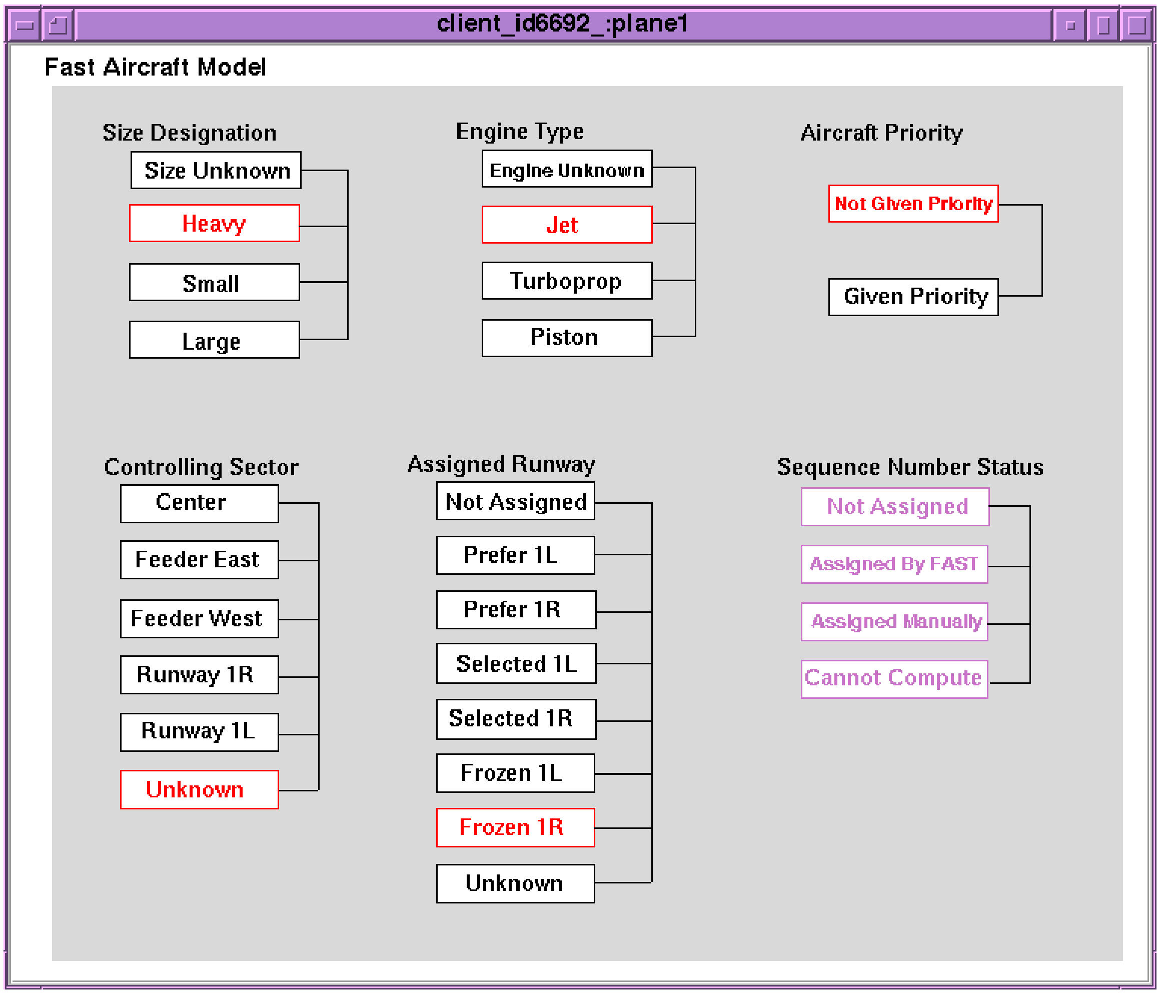


Figure 28: Visualization of the FAST Aircraft Model

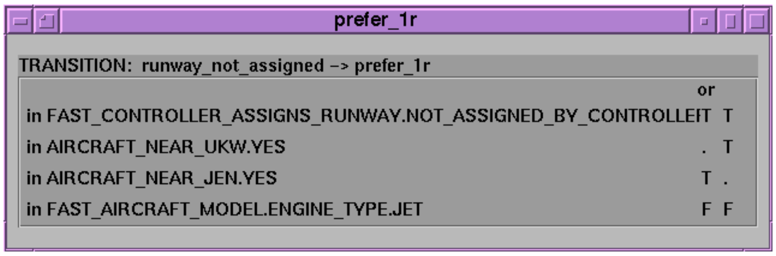
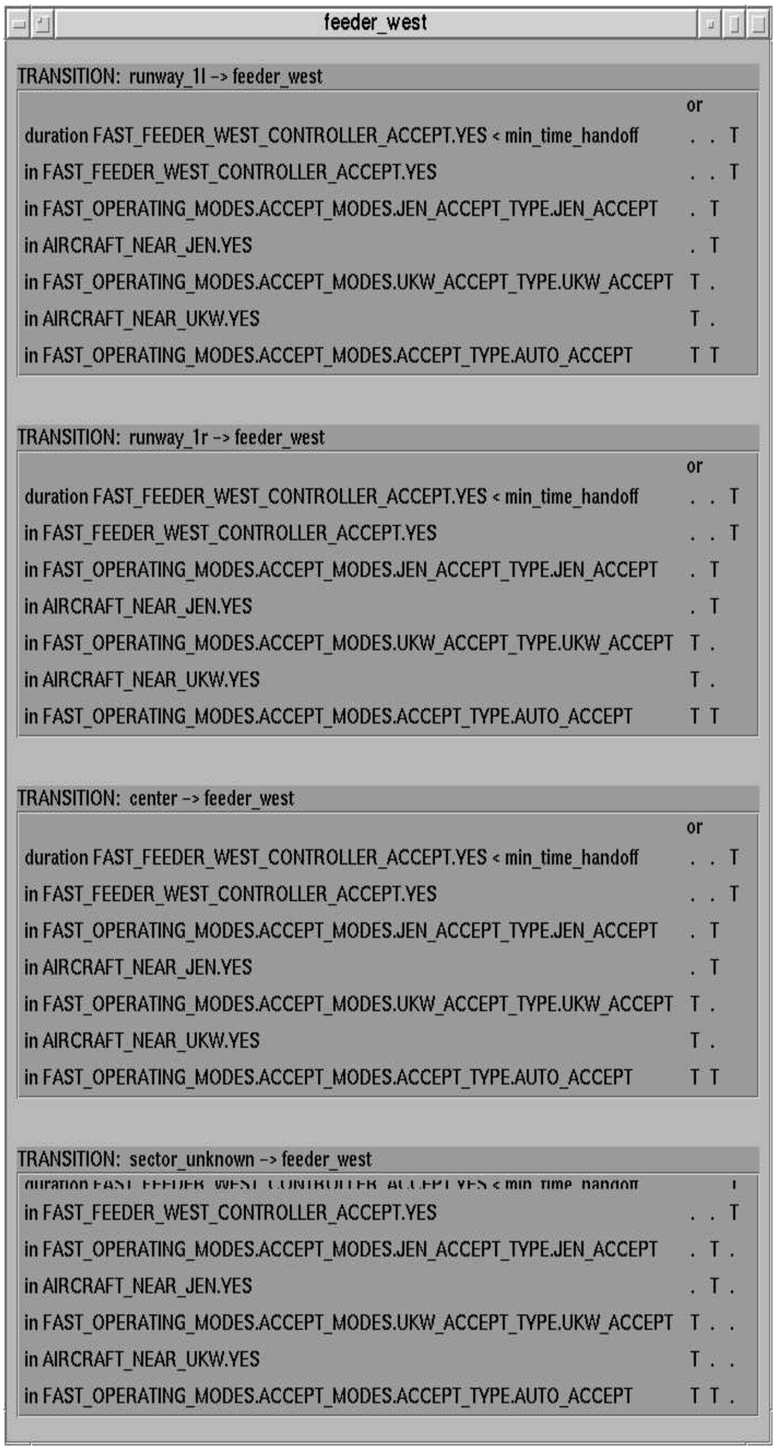


Figure 29: Example transition into the state $Prefer_1R$


feeder_west

TRANSITION: runway_1l -> feeder_west

	or		
duration FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES < min_time_handoff	.	.	T
in FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES	.	.	T
in FAST_OPERATING_MODES.ACCEPT_MODES.JEN_ACCEPT_TYPE.JEN_ACCEPT	.	T	
in AIRCRAFT_NEAR_JEN.YES	.	T	
in FAST_OPERATING_MODES.ACCEPT_MODES.UKW_ACCEPT_TYPE.UKW_ACCEPT	T	.	
in AIRCRAFT_NEAR_UKW.YES	T	.	
in FAST_OPERATING_MODES.ACCEPT_MODES.ACCEPT_TYPE.AUTO_ACCEPT	T	T	

TRANSITION: runway_1r -> feeder_west

	or		
duration FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES < min_time_handoff	.	.	T
in FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES	.	.	T
in FAST_OPERATING_MODES.ACCEPT_MODES.JEN_ACCEPT_TYPE.JEN_ACCEPT	.	T	
in AIRCRAFT_NEAR_JEN.YES	.	T	
in FAST_OPERATING_MODES.ACCEPT_MODES.UKW_ACCEPT_TYPE.UKW_ACCEPT	T	.	
in AIRCRAFT_NEAR_UKW.YES	T	.	
in FAST_OPERATING_MODES.ACCEPT_MODES.ACCEPT_TYPE.AUTO_ACCEPT	T	T	

TRANSITION: center -> feeder_west

	or		
duration FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES < min_time_handoff	.	.	T
in FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES	.	.	T
in FAST_OPERATING_MODES.ACCEPT_MODES.JEN_ACCEPT_TYPE.JEN_ACCEPT	.	T	
in AIRCRAFT_NEAR_JEN.YES	.	T	
in FAST_OPERATING_MODES.ACCEPT_MODES.UKW_ACCEPT_TYPE.UKW_ACCEPT	T	.	
in AIRCRAFT_NEAR_UKW.YES	T	.	
in FAST_OPERATING_MODES.ACCEPT_MODES.ACCEPT_TYPE.AUTO_ACCEPT	T	T	

TRANSITION: sector_unknown -> feeder_west

duration FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES < min_time_handoff			T
in FAST_FEEDER_WEST_CONTROLLER_ACCEPT.YES	.	.	T
in FAST_OPERATING_MODES.ACCEPT_MODES.JEN_ACCEPT_TYPE.JEN_ACCEPT	.	T	.
in AIRCRAFT_NEAR_JEN.YES	.	T	.
in FAST_OPERATING_MODES.ACCEPT_MODES.UKW_ACCEPT_TYPE.UKW_ACCEPT	T	.	.
in AIRCRAFT_NEAR_UKW.YES	T	.	.
in FAST_OPERATING_MODES.ACCEPT_MODES.ACCEPT_TYPE.AUTO_ACCEPT	T	T	.

Figure 30: Example of transition tables for transitions into the state *FeederWest*

7 State Machine Hazard Analysis

Top-down and backward hazard analyses start from a hazard and determine if and how the hazardous state can be reached. We have found that the backward reachability graph explodes quickly even though many of the branches are physically impossible. Therefore, we currently implement the process in SpecTRM by having the analyst start the model in a hazardous state and work back one step at a time, using our backward simulation capability. At each step, the analyst prunes the tree of irrelevant branches and decides which branch to follow next.

Using this process, we can implement the safety analysis algorithms defined by Leveson and Stolzy [LS87] that evaluate whether the system can reach a hazardous state when operated as specified and when there are various types of failures. Complete reachability analysis, even going backward from one state, is infeasible for large state-machine models. However, the analysis algorithms are designed to detect the *critical states* on the paths to hazardous states. These critical states are informally defined as states from which there are paths that lead to the hazardous state and paths that lead to other states. The identification of a critical state can be used to change the design in a way that eliminates the path to the hazardous state, or, if that is not possible, to design suitable controls. The critical state itself might not be reachable, but that only means that the procedures will eliminate hazardous states that could not really have been reached.

We performed this type of hazard analysis on the DFW model with only limited results. Part of the reason revolves around our switch from RSML to our new modeling language, SpecTRM-RL, midstream. We began developing a specification of CTAS in RSML, but our new visualization tools were developed to work with SpecTRM-RL models so the effort was switched to developing a model in the new language. Unfortunately, we discovered that we did not have enough time to switch our older analysis tools (backward simulation, consistency and completeness checker, and deviation analysis) to the new language. Therefore, we ran them on the partially specified versions of the RSML model of CTAS. Although for completeness and consistency checking this change simply provided us with more instances of incompleteness to detect, the state machine hazard analysis and deviation analysis tools provided only limited results (the RSML models were too simple to contain interesting hazards), and therefore the results are only suggestive of how the techniques would work on a less superficial CTAS specification. For example, we found that two different menus could possibly be popped up at the same time (in our RSML model) when they were supposed to be mutually exclusive. Previous use of these tools on more complete models, such as a flight management system, produced more interesting results [MLRPS97].

8 Deviation Analysis and FMECA

Failure Modes and Effects Analysis (FMEA) and Failure Modes and Effects Criticality Analysis (FMECA) are most effective in examining the effects of hardware-related failures. A limitation of the approach is that it usually looks at single failures only, although accidents in complex systems often occur as a result of multiple events or failures.

Applying a FMEA or any bottom-up or forward analysis technique to software is complicated by the large number of ways that computers can contribute to system hazards. A valve that has only two or three relevant discrete states (such as open, closed, or partially open) and a similarly limited number of failure states (fail open, fail shut, or fail partially open), for example, can be examined for the potential effects of these states on the system state. Computers, however, can assume so many states, exhibit so many visible and potentially important behaviors, and have such a complex effect on the system that complete bottom-up or forward system analyses are, in most cases, impractical. In simpler systems, where the software can affect only a few system parameters, such an analysis might be useful.

FMEA and FMECA can provide important information, however, and they should be performed for systems where hardware failure (such as radars) is possible. They are most helpful in determining what effects complete failure of major components of the system might have. In the case of CTAS, the changes involved in using FAST do not have a major impact on the backup procedures currently in place.

We examine the problem of erroneous outputs from FAST elsewhere in this report, but the total failure of the CTAS software or computer hardware should not, in general, be any different than a current failure of the RDP. Such a failure might have an important impact if the use of the automation allowed less spacing and more aircraft to be handled by a single controller than is true today. In that case, the necessity of the controller to handle increased traffic using flight strips alone might lead to a hazard. Although this problem does not seem to exist for passive FAST, the operational system should be audited periodically to make sure that the automation is not used in an unexpected way to increase controller load beyond what could be handled safely using current manual backup procedures.

A detailed analysis of the failure modes and effects related to total failure of CTAS will depend upon details of the installation at DFW, to which we have no access. For example, the effect of CTAS failure may depend on whether the TRACON has a DARC or EDARC system. DARC is the Direct Access Radar Channel, while EDARC is enhanced DARC. DARC/EDARC is, in essence, an independent path (and a much different version of digital logic) from the radar input lines at the centers to the controllers, although there are still several common points of failure with the main RDP path. At the enroute centers, the controllers are expected to respond to a failure of the RDP by switching their displays (PVDs) to EDARC. They will not see "raw" radar returns, but the data is a lot less processed than normal RDP data. So the results of a FMECA would depend on whether the DFW TRACON has DARC, if it is planned to be retained with CTAS, and if the new displays have the capability to display it.

In addition, if the FAST logic is implemented as a coroutine to coresident RDP logic, FAST can probably preclude the proper functioning of RDP in several ways. Even if FAST is independent of RDP, there are a couple of cases where FAST could bring down RDP.

In general, the use of forward analyses, like FMEA and FMECA or HAZOP, run into difficulties when applied to complex systems with software components. When the tracing of a failure (FMECA) or deviation (HAZOP) reaches a computer component, it may be difficult to determine what affect that failure will have on the software behavior and outputs, particularly before the software has been implemented. Even after it is implemented, only a limited number of test cases can be evaluated. We solve this problem using a new forward analysis technique for software developed by Reese [Ree96, Ree96] called Software Deviation Analysis (SDA).

Like HAZOP, SDA is based on the underlying assumption that many accidents are the result of deviations in system variables. A deviation is the difference between the actual and correct values. SDA can determine whether a hazardous software behavior (usually an output) can result from a class of input deviations, such as measured aircraft speed too low (the measured or assumed speed is less than the actual speed). SDA is a way of evaluating system components for *robustness* (in the security community this is often called *survivability*) or how they will behave in an imperfect environment.

Figure 31 shows an overview of the procedure. The analyst provides a formal software requirements specification (e.g., an RSML or SpecTRM-RL specification), which the procedure automatically converts into a *causality diagram*. The causality diagram is an internal data structure that encodes causal information between system variables, based on the specification and the semantics of the specification language. The simplicity of causality diagrams makes the search algorithm more straightforward and easier to adapt to a new specification language. Causality diagrams may also be helpful to the analyst in understanding how system variables are inter-related.

At this point, the causality diagram describes the relationship between the actual values of system variables, but not their deviations. The procedure next augments the causality diagram with *deviation formulas* so that variable deviations are represented.

After the analyst's specification is converted to an augmented causality diagram, the procedure is ready for the analyst to identify the safety-critical software outputs and at least one deviation in the input environment.

The augmented causality diagram, safety-critical software outputs, and the initial deviations are passed to the search program, which constructs a tree of states. The initial deviations are at the root of the search tree. Leaves are either dead-end searches (in which the state does not contain any deviations) or states containing safety-critical deviations.

The search is based on applying *qualitative mathematics* to the causality diagram. Qualitative mathematics partitions infinite domains into a small set of intervals and provides mathematical operations on these intervals. The use of fixed intervals simplifies the analysis compared to iterations over the entire state space. Like HAZOP, it also lends itself naturally to the qualitative nature of deviations, such as "slightly too high."

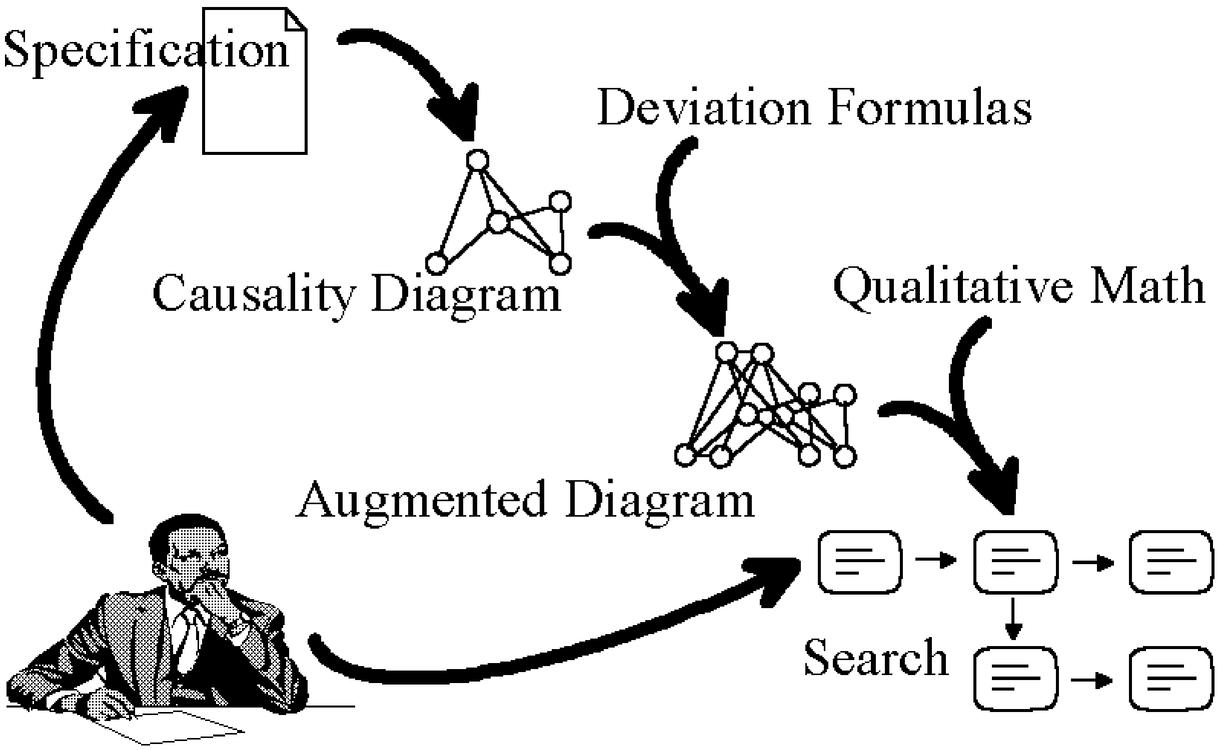


Figure 31: Overview of deviation analysis procedure.

The output of the procedure is a list of *scenarios*. A scenario is a set of deviations in the software inputs plus constraints on the execution states of the software that are sufficient to lead to a deviation in a safety-critical software output. The deviation analysis procedure can optionally add further deviations as it constrains the software state, allowing for the analysis of multiple independent failures.

We did not perform a FMECA or HAZOP for this study; the process is very straightforward. Instead, we demonstrated how SDA could be used to provide information for a FMECA or other forward analysis. Unfortunately, we again ran into the problem that the conversion of our SDA tool from RSML to SpecTRM-RL was not completed in time to apply SDA to the more complete SpecTRM-RL model. We were able to run the tool, but we did not find any serious problems in our simple model. Figures 32 and 33 show the input and output (respectively) from a sample execution of our SDA tool. We are currently working on better ways to visualize the output from the deviation analysis.

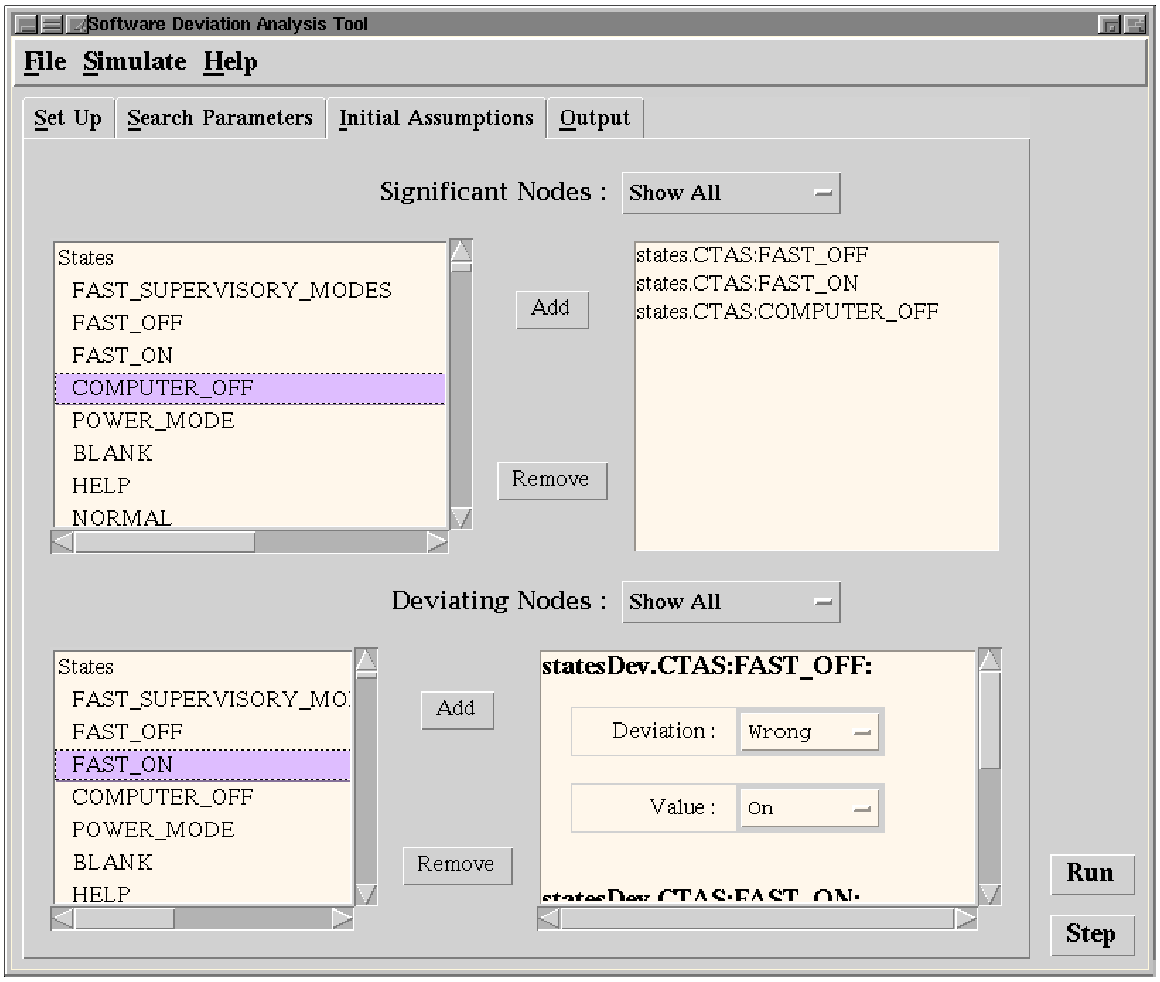


Figure 32: Deviation analysis input screen

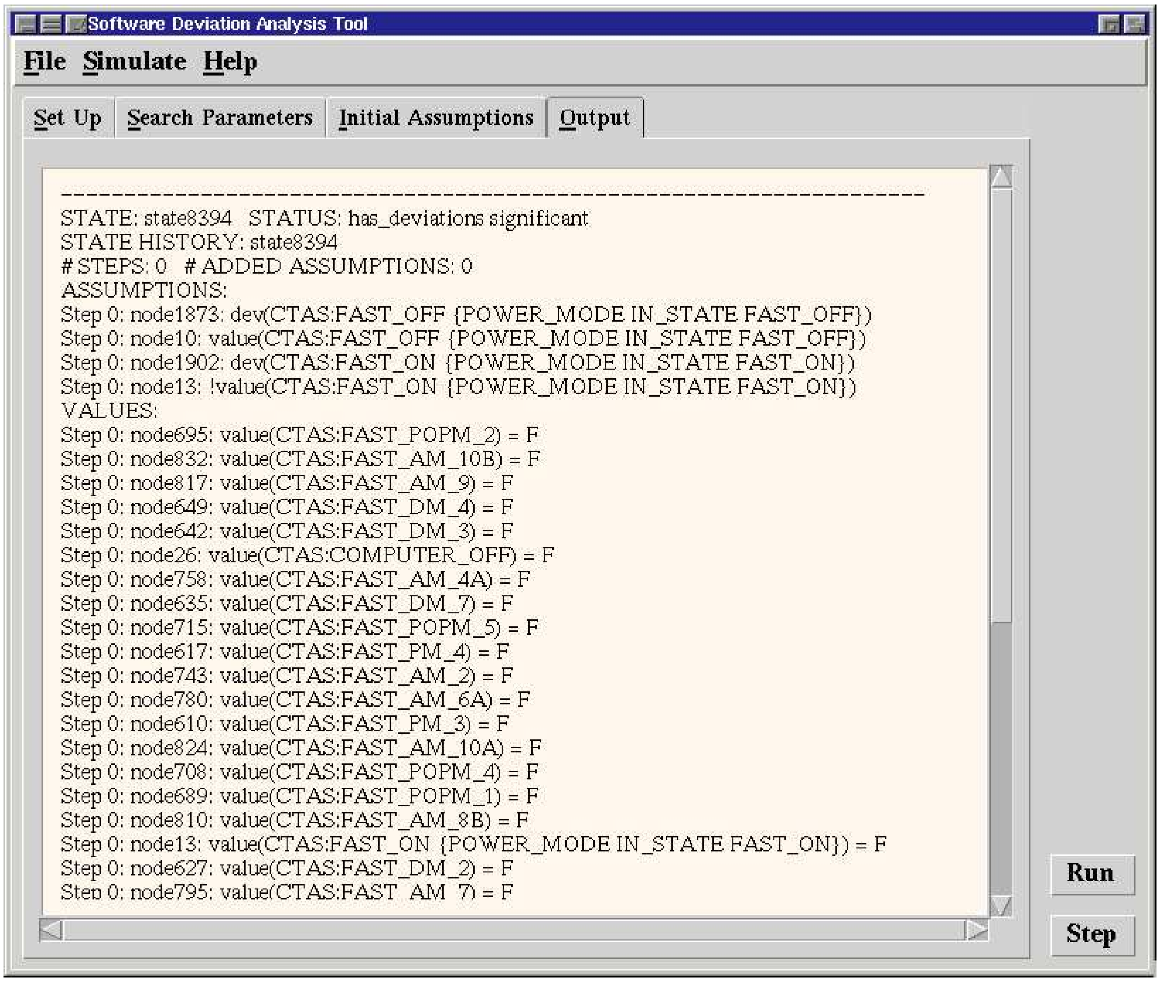


Figure 33: Deviation analysis output

9 Mode Confusion Analysis

As the role of automation changes in a system, so does the role of human operators. A safety analysis must examine these changes for their potential effect on human error that could lead to accidents and identify ways to change the system design and operator training to reduce this potential. In this section and the next, we describe two complementary approaches to this problem: The first focuses on the automation to evaluate its potential to contribute to human error while the second focuses more on the human to evaluate the effect of the proposed system changes on human performance (see Section 10).

Increased automation in complex systems has led to changes in the human controller's role and to new types of technology-induced human error. Some of these errors are the result of what Sarter, Woods, and Billings have called *technology-centered automation* [SW95a]. Too often, the designers of the automation focus on technical aspects and do not devote enough attention to the cognitive and other demands on the operator. The result of technology-centered automation has been what Wiener calls *clumsy automation*. Software engineers building embedded controllers are rarely taught or understand the set of cognitive processing activities associated with maintaining situation and mode awareness and how their designs can affect these human activities. Instead, they tend to focus on the mapping from software inputs to outputs, on mathematical models of required functionality, and on the technical details and problems internal to the computer. Little attention has been given to evaluating software in terms of whether it provides transparent and consistent behavior that supports operators in their monitoring and control tasks.

In our examination of CTAS, we were pleased to see the amount of thought that had gone into the human-machine interaction component of the system. But passive FAST and even active FAST are currently limited in their control activities and primarily provide information to the controller. In order to achieve the increased throughput in the ATC system that is planned for the future, automation is going to have to take over more direct control functions. It is at this point, where the system is being controlled jointly by computers and humans, that other complex systems have started to experience accidents related to a lack of coordinated activities and information flow between the various controllers. One particularly problematic feature of these new designs is a proliferation of modes, where (as defined earlier) modes define mutually exclusive sets of system behavior. Examples of operating modes relevant to CTAS are waypoint-capture mode and route-intercept mode.

Mode-rich systems provide flexibility and enhanced capabilities, but they also increase the need for and difficulty of maintaining mode awareness, which can lead to new types of mode-related problems. Attempts to mitigate these errors have primarily involved giving more authority to the automation, enhancing operator training, or changing the interface. Giving more authority to the automation has led to serious accidents that operators tried to prevent but could not because of their authority limits.

Eliminating humans completely from these control loops is simply not possible at this time while ensuring an acceptable level of safety. Therefore, humans will continue to be in these control loops, and, if it is true that mode-related human errors are caused by clumsy or poorly designed automation, then changing the human interface, operator training, or operational procedures is not the only possible solution to our problems: "Training cannot and should not be the fix for bad design" [SW95a]. Instead, if we can identify automation design characteristics that lead to mode awareness errors or that increase cognitive demands, then we may be able to redesign the automation without reducing system capabilities. In addition, knowing the causes of increased cognitive load will make changes in training or interface design more effective. The approach chosen will depend upon such factors as relative costs, perceived effectiveness, and required tradeoffs.

This section describes an approach to detecting error-prone automation features early in the development process while significant changes can still be made to the conceptual design of the system [LPS97, LP97]. To accomplish this goal, designers need to be able to identify problematic design features. Our approach to this problem is to identify design constraints on the automation based on known cognitive constraints on the human operator and engineered or natural environmental constraints. These constraints are based on the types of errors that humans make in highly automated systems, as determined by human factors studies and by examining past accidents in highly automated systems. Using this information, we analyze the blackbox behavior specified in our blackbox models of the automation (the SpecTRM-RL models) to predict where errors will occur and use this information to design the automation; to design or evaluate the design of the operator procedures, tasks, and interface; and to make tradeoff decisions in the early development stages of the system. In fact, part of the reason for modeling modes the way we do in SpecTRM-RL is based on our desire to analyze the models for mode-confusion potential.

For our analysis approach to work, human errors must be non-random. After studying accidents and incidents in the new, highly automated aircraft, Sarter and Woods [SW95b] have concluded that certain errors are predictable: They are the regular and predictable consequences of a variety of identifiable factors. Although they are "accentuated" by poor interface design and gaps or misconceptions in the user's mental model of the system, mismatches between expected and actual automation behavior is not necessarily related to an inadequate operator mental model, but can also result from inconsistent automation behavior. Sarter and Woods identify some of these error forms.

In the rest of this section, we first define mode confusion more carefully and then define some design and analysis criteria we have identified to detect features of blackbox state-based requirements specifications (i.e., features of SpecTRM-RL models) that are likely to lead to mode-confusion errors.

9.1 Mode Confusion

High-tech systems, such as aircraft control systems, are starting to experience a proliferation of modes. The new mode-rich systems provide flexibility and enhanced capabilities, but they also increase the need for and difficulty of maintaining mode awareness, which can lead to new types of mode-related problems.

While automation has eliminated some types of mode-awareness errors, it has also created the potential for new types of errors. Sarter and Woods [SW95a] extend the classic definition of mode error and distinguish between errors of *commission* (where an operator takes an inappropriate action) and errors of *omission* (where the operator fails to take a required action).

The first automated systems tended to have only a small number of independent modes, and functions were associated with one overall mode setting. In addition, the consequences of operator mode awareness problems tended to be minor, partly because feedback about operator errors was fast and complete enough that operators were able to recover before the errors caused serious problems (Rasmussen's concept of *error tolerance* [Ras90]).

Studies of less complex aircraft automation show that pilots sometimes lose track of the automation behavior and experience difficulties in directing the automation, primarily in the context of highly dynamic and/or non-normal situations [SW95a]. In most cases, these problems are associated with errors of *commission*, that is, with errors that require a pilot action in order for the problem to occur. This type of error is the classic mode error identified and defined by Norman—an intention is executed in a way that is appropriate for one mode but the device is actually in a different mode. Because the operator has taken an explicit action, he or she is likely to check that the intended effect of the action has actually occurred. The short feedback loops allow the operator to repair most errors before serious consequences result. This type of error is still the prevalent one on relatively simple devices such as word processors.

In contrast, studies of more advanced automation in aircraft like the A-320 find that mode errors of *omission* are the dominant form of error [SW95b]. In this type of mode error, the operator *fails* to take an action that is required, perhaps because the automation has done something undesirable (perhaps involving a mode change) and the operator does not notice. In other words, the operator fails to detect and react to an undesired system behavior that he or she did not explicitly invoke. Because the mode or behavioral changes are not expected, the operator is less likely to pay attention to the relevant indications (such as mode annunciations) at the right time and detect the mode change or undesired behavior.

Errors of omission are closely related to the role change of the operator from direct control to monitor, exception handler, and supervisor of the automation. As these roles change, the operator tasks and cognitive demands are not necessarily reduced, but instead tend to change in their basic nature. The added or changed cognitive demands tend to congregate at high-tempo, high-criticality periods [SW95a]. While some types of errors and failures have declined, new error forms and paths to system breakdown have been introduced.

Some of these new error forms are a result of mode proliferation without appropriate support. Providing support has been complicated by some unexpected changes in operator behavior in working with complex automation. For example, during long periods of flight, pilots do not have to monitor the mode annunciations continuously. Instead, they need to predict the occurrence of mode transitions in order to attend to the right indications at the right time. A-320 pilots have identified this new type of monitoring behavior in surveys conducted by Sarter and Woods. However, the automation and interfaces have been designed assuming conventional monitoring.

Simply calling for systems with fewer or less complex modes is unrealistic: Simplifying modes and automation behavior often requires tradeoffs with increased precision or efficiency and with marketing demands from a diverse set of customers [SW95a]. However, systems may exhibit *accidental complexity* where the automation can be redesigned to reduce the potential for human error without sacrificing system capabilities. Where tradeoffs with desired goals are required to eliminate potential mode-confusion errors, hazard analysis may be able to assist in providing the information necessary for appropriate decision making.

In section 5 we described completeness criteria we have identified for behavioral specifications of control systems. In attempting to extend these criteria (design constraints) to cover mode-confusion errors, we built on the experience accumulated by human factors experts such as the results of Sarter and Woods' studies of A-320 accidents and incidents along with other reports of mode-related error. We have so far identified six categories of potential design flaws: interface interpretation errors, inconsistent behavior, indirect mode changes, operator authority limits, unintended side effects, and lack of appropriate feedback.

Before describing each of these, we note that applying our criteria to a complex control system will almost surely identify a large number of behaviors that could lead to mode confusion. Getting rid of all such behaviors would most likely result in an overly simple control system that does not satisfy many of its goals. Instead, this information should be used to redesign the automation, to provide information for safety-related tradeoff decisions, and to design interfaces, operational procedures, and operator training programs. For example, accidents most often occur during transitions between normal and non-normal operating modes or while operating in non-normal modes. Therefore, the non-normal mode transitions should be identified and have more stringent design constraints applied to them.

9.2 Interface Interpretation Errors

Interface mode errors are the classic form of mode confusion error: (1) the computer interprets user-entered values differently than intended or (2) it maps multiple conditions onto the same output depending on the active operational mode and the operator interprets the interface erroneously.

A common example of an input interface interpretation error occurs with many 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.

An example of an output interface mode problem was identified by Cook et.al. [CPWM91] in a medical operating room device with two operating modes: warmup and normal. The device starts in warmup mode when turned on and changes from normal mode to warmup mode whenever either of two particular settings are adjusted by the operator. The meaning of alarm messages and the effect of controls are different in these two modes, but neither the current device operating mode nor a change in mode are indicated to the operator. In addition, four distinct alarm-triggering conditions are mapped onto two alarm messages so that the same message has different meanings depending on the operating mode. In order to understand what internal condition triggered the message, the operator must infer which malfunction is being indicated by the alarm.

A more complex example occurs in a proposed A-320 accident scenario where the crew directed the automated system to fly in the TRACK/FLIGHT PATH ANGLE mode, which is a combined mode related to both lateral (TRACK) and vertical (FLIGHT PATH ANGLE) navigation:

> When they were given radar vectors by the air traffic controller, they may have switched from the TRACK to the HDG SEL mode to be able to enter the heading requested by the controller. However, pushing the button to change the lateral mode also automatically changes the vertical mode from FLIGHT PATH ANGLE to VERTICAL SPEED—the mode switch button affects both lateral and vertical navigation. When the pilots subsequently entered "33" to select the desired flight path angle of 3.3 degrees, the automation interpreted their input as a desired vertical speed of 3300 ft. This was not intended by the pilots who were not aware of the active "interface mode" and failed to detect the problem. As a consequence of the too steep descent, the airplane crashed into a mountain [SW95b].

Several design constraints can assist in reducing interface interpretation errors. The first 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 our SpecTRM-RL modeling language). More generally, the current operating mode of the automation should be annunciated (should be in the displays interface). In addition, any change of operating mode should trigger a change in the current operating mode reflected in the interface (and thus displayed to the operator), i.e., the annunciated mode must be consistent with the internal operating mode. Consistency between displayed and current mode is, of course, an obvious design constraint and a violation almost always signals an error in the requirements specification. The first constraint should hold for almost all systems as well.

Degani [Deg96] notes a third type of interface confusion error that results from mapping a single input control action to multiple internal mode changes, depending on the order of the control actions. He calls this *circular mode transitions.* For example, pushing a button on a device with a small input interface (e.g., a watch with one or two buttons) will often cycle through the possible modes, going to the next mode with the next button push. The user can get confused about what input mode the device is currently in. A possible design constraint here is that if a control input is used to trigger a mode transition, then it must be associated with only one mode change, that is, the mapping from control inputs to mode changes is one-to-one (a mathematical function). Note that it is unlikely that one would want to require that the function be bijective, because that would eliminate the possibility of all indirect mode changes. For some simple devices, even the constraint that the function be injective (one-to-one) may be impossible to enforce, and feedback about the current mode is the only possible solution to the problem.

Another design constraint related to these types of interface interpretation errors is that interpretation of the supervisory interface should not be conditioned on modes (an example is the accident related to the interpretation of "33" described earlier). This constraint is much stronger than the first three and may not always be feasible or desirable to enforce. However, our analysis tools will highlight these transitions to the designer/analyst so that appropriate scrutiny can be applied to that part of the design. Degani's circular mode transition is a subcase of this design constraint.

9.3 Inconsistent Behavior

A more complex type of mode-confusion error, which is more often related to errors of omission than the interface errors mentioned above, is triggered by inconsistent behavior of the automation. Carroll and Olson define a consistent design as one where a similar task or goal is associated with similar or identical actions [CO88]. Consistent behavior makes it easier for the operator to learn how a system works, to build an appropriate mental model of the automation, and to anticipate system behavior.

An example of inconsistency was detected in an A-320 simulator study involving a go-around below 100 feet above ground level. Sarter and Woods found that pilots failed to anticipate and realize that the autothrust system did not arm when they selected TOGA (take off/go around) power under these conditions because it did so under all other circumstances where TOGA power is applied [SW95b]. Another example of inconsistent automation behavior, which was implicated in an A-320 accident, involves a protection function that is provided in all automation configurations except the altitude acquisition mode in which the autopilot was operating.

Consistency is particularly important in high-tempo, highly dynamic phases of flight where pilots may have to rely on their automatic systems to work as expected without constant monitoring. Even in more low pressure situations, consistency (or predictability) is important in light of the evidence from pilot surveys that their normal monitoring behavior may change on advanced flight decks [SW95b].

Pilots on conventional aircraft use a highly trained instrument scanning pattern of recurrently sampling a given set of basic flight parameters. In contrast, some A-320 pilots explained that they no longer have a scan anymore but allocate their attention within and across cockpit displays on the basis of expected behavior. Their monitoring objective is to verify expected automation states and behaviors. If the automation behavior is not consistent, mode errors of omission may occur where the pilot fails to intervene when necessary:

> Note the fundamental difference between these two monitoring strategies. In the case of a standard pattern, the pilot's attention allocation is externally

guided while monitoring on advanced aircraft requires mental effort on the part of the pilot who has to determine on his own where to look next under varying task circumstances. Based on his expectations, the pilot only monitors part of all available data. Parameters that are not expected to change may be neglected for a long time. A standard instrument scan, on the other hand, serves to ensure that all relevant parameters concerning airplane behavior will be monitored at certain time intervals to make sure that no unexpected and maybe undesirable changes occur [SW95b].

In our previous design criteria and analysis tools, we include a check for nondeterminism in the software behavior; that is, we check to determine whether more than one transition can be taken out of a state under the same conditions [HL96]. But consistency in this case requires more than simple deterministic behavior on the part of the automation. If the operator provides the same inputs but different outputs (behaviors) result for some reason other than what the operator has done (or even may know about), then the behavior is inconsistent from the operator viewpoint even though it is not mathematically inconsistent or nondeterministic. More formally, inconsistent behavior results from two state transition functions of the form:

$$t_1 : s \times i_o \times x \to \{s \times O\}'$$
$$t_2 : s \times i_o \times y \to \{s \times O\}''$$

where $s \in \Sigma$ is a state, i_o is an operator input, O is an output, and x and y can be states, reference values, supervisory interface values, etc.

We have identified several different design constraints related to various types of inconsistency. However, there may be reasons why having such inconsistencies is necessary or reasonable. Again, our tools can point out such potential problems to the designer/analyst who must make the final decision about whether the automation should be changed. Because consistency may be most important during critical situations or when the behavior is related to a safety design constraint, our hazard analysis tools may be able to assist with these decisions and our new intent specifications [Lev97] (a form of Rasmussen's means-ends hierarchy adapted for software), as described in Section 12, can be used to trace such behavior back to its original system goals and safety constraints to identify any reasons for the specified inconsistent behavior.

9.4 Indirect Mode Changes

Indirect mode changes occur when the automation changes mode without an explicit instruction by the operator. Such transitions may be triggered on conditions in the automated controller (such as preprogrammed envelope protection) or sensor input about the state of the controlled system (such as achievement of a preprogrammed target or an armed state with a preselected mode transition).

Like many of the other mode-confusion problems noted in this paper, indirect mode transitions create the potential for mode errors of omission and of inadvertent activation of modes by the operator. Again, the problems are related to changes in scanning methods and difficulty in forming expectations of uncommanded or externally triggered behavior.

Behavioral expectations are formed based on the operators' knowledge of input to the automation and on his or her mental model of the automation's designed behavior. Gaps or misconceptions in the operator's mental model may interfere with predicting and tracking indirect mode transitions or with understanding the interactions between different modes.

An example of an accident that has been attributed to an indirect mode change occurred while an A-320 was landing in Bangalore. In this case, the pilot selection of a lower altitude while the automation was in the ALTITUDE ACQUISITION mode resulted in the activation of the OPEN DESCENT mode. It has been speculated that the pilots did not notice the mode annunciation because the indirect mode change occurred during approach when the pilots were busy and they were not expecting the change [SW95a]. Another example of such an indirect mode change in the A-320 automation involves an automatic mode transition triggered when the airspeed exceeds a predefined limit. For example, if the pilot selects a very high vertical speed that results in the airspeed decreasing below a particular limit, the automation will change to the OPEN CLIMB mode, which allows the airplane to regain speed. As a final example, Palmer has described an example of a common indirect mode transition problem called a "kill-the-capture bust" that has been noted in hundreds of ASRS reports [Pal96]. Leveson and Palmer have modeled an example of this problem in SpecTRM-RL and shown how it could be detected and fixed [LP97].

In general, there are four ways to trigger a mode change:

1. Operator explicitly selects a new mode.
2. Operator enters data (such as a target altitude) or a command that leads to a mode change:
   (a) Under all conditions.
   (b) When the automation is in a particular state.
   (c) When the controlled system model or environment is in a particular state.
3. Operator does not do anything but the transition is triggered by conditions in the controlled system.
4. Operator selects a mode change but the automation does something else, either because of the state of the automation and/or the state of the controlled system.

Operator errors associated with indirect mode changes are a phenomenon found primarily in advanced automation. Early automation tended to involve only a small number of independent modes. Most functions were associated with only one overall mode setting. We probably do not want to go back to automation that will change mode only in response to direct operator input, but design constraints are desirable that limit such indirect transitions and eliminate it when possible. Our analysis methods highlight mode changes that are independent of direct and immediate instructions from human supervisors, and our tools may also be able to assist the analyst in identifying the most hazardous indirect mode changes.

9.5 Operator Authority Limits

Interlocks and lockouts are often used to ensure safety. *Interlocks* are commonly used to prevent hazardous system states by enforcing correct sequencing of events or actions or to isolate two events in time. A *lockout* makes it impossible or difficult to enter a hazardous state.

Authority limiting is a type of lockout or interlock that prevents actions that could cause the system to reach hazardous states. Such authority limitations must be carefully analyzed to make sure they do not prohibit maneuvers that may be needed in extreme situations. Recent events have involved pilots "fighting" with the automation over control of the aircraft after observing unexpected or undesirable aircraft or automation behavior.

Various types of authority limits are used to prevent operator error or to provide protection when the operator cannot or does not take proper action. For example, advanced aircraft automation often has the ability to detect and prevent or recover from predefined unsafe aircraft configurations such as a stall. Once a hazardous state is detected, the automation has the power to override or limit pilot input.

Some accidents and incidents in highly automated aircraft have involved pilots not being able to overcome the protection limits or the pilots not being aware that the protection functions were in force. For example, the pilots during one A-320 approach disconnected the autopilot while leaving the flight directors and the autothrust system engaged. Under these conditions, the automation provides automatic speed protection by preventing the aircraft from exceeding upper and lower airspeed limits:

> At some point during the approach, after flaps 20 had been selected, the aircraft exceeded the upper airspeed limit for that configuration by 2 kts. As a consequence, the automation intervened by pitching the airplane up to reduce airspeed back to 195 kts. The pilots, who were not aware that the automatic speed protection was active, observed the uncommanded automation behavior. Concerned about the unexpected reduction in airspeed at this critical phase of flight, they rapidly increased thrust to counterbalance the automation. As a consequence of this sudden burst of power, the airplane pitched up to about 50 degrees, entered a sharp left bank, and went into a dive. The pilots eventually disengaged the autothrust system and its associated protection function and regained control of the aircraft [SW95b].

Various design criteria are related to authority limits. For example, information about any modes or states where the operator input is ignored or limited must be provided in the supervisory interface. In addition, the analysis tools can examine the specified software behavior and detect exceptions to following operator requests. Again, the information in the intent specification is useful in determining whether such design features are intentional and whether they are related to identified hazards.

9.6 Unintended Side Effects

Mode ambiguity can also arise when an action intended to have one particular effect has an additional effect, i.e. an unintended side effect. An example occurred in a Sarter and Woods A-320 simulator study where it was discovered that pilots were not aware that entering a runway change *after* entering data for the assigned approach results in the deletion of all previously entered altitude and speed constraints even though they may still apply.

This type of design flaw differs from indirect mode changes in that the unintended change is not in the mode but in some other type of information, such as reference values. Degani describes this type of problem in terms of a mode/reference value interaction, but more generally the same problem occurs when any operator entry (for example, an input value rather than a mode change) has unintended side effects.

Unintended side effects can contribute to mode confusion and often need to be evaluated by the design team. If a decision is made to keep the behavior, proper feedback constraints may be required to prevent the type of confusion that seems to result.

9.7 Lack of Appropriate Feedback

Many of the original Jaffe criteria or the newly defined criteria mentioned above are related to providing appropriate feedback (for example, providing feedback about the status of interlocks and lockouts and providing graceful degradation). In general, operators need to have the information necessary to understand the mode transitions taken, that is, the conditions that trigger transitions between modes. Operators need not only to track the current active modes and to understand their implications, but they also need to keep track of other automation and system status information that may result in the indirect activation of modes. The difference between these design constraints and those requiring mode transition annunciations described in the section on interface interpretation errors is that in this case the automated system must not simply notify the operator that a mode change has already occurred (annunciate the present mode), but it must provide the information necessary for the operator to *predict* or *anticipate* mode changes.

Incomplete feedback is often implicated in accident scenarios. For example, in the A-320 Bangalore accident, the pilot flying (PF) had disengaged his flight director during the approach and was assuming that the pilot-not-flying (PNF) would do the same thing [SW95a]. The result would have been a mode configuration in which airspeed is automatically controlled by the autothrottle (the SPEED mode), which is the recommended procedure for the approach phase. However, the PNF never turned off his flight director, and the OPEN DESCENT mode became active when a lower altitude was selected. This indirect mode change (explained above) led to the hazardous state and eventually the accident. But a complicating factor was that each pilot only received an indication of the status of his own flight director and not all the information necessary to determine whether the desired mode would be engaged. The lack of feedback or knowledge of the complete system state contributed to the pilots not detecting the unsafe state in time to reverse it.

Where automation has the ability to take autonomous actions (i.e., those not directly commanded by the operator), information interchange becomes crucial in coordinating activities and in detecting mismatches between expected and actual system behavior. A behavioral description of the software, as provided by SpecTRM-RL, is useful in determining exactly what information the operator needs to monitor and control the automated system.

The problems of providing salient feedback are, of course, much more complicated than simply identifying the information that needs to be conveyed, but identification is an important step in the process. In our original Jaffe criteria (Appendix D), we identified design constraints on basic feedback to the computer about the state of the controlled process and some types of operator feedback requirements, but these need to be augmented with a complete set of requirements on the feedback to the operator or automation supervisor. An example constraint is that operators must have access to all information on critical mode transitions in order to predict and monitor those transitions.

One important aspect of using feedback for error detection is the need for independent information. Errors can only be found through discrepancies in duplicate but independently supplied information. One way to detect that automated equipment is not operating correctly is for operators to detect a discrepancy between the automation behavior and their mental model of how they think the automation should work. However, operators often have limited understanding of complex automation behavior or are afraid to step in.

In addition, often an error is only detectable using some information about the state of the environment or the controlled process. However, if the erroneous behavior is occurring because the automation is confused about the environment or system state, then it obviously cannot provide this information to the operator. That is, the automation may show only consistent information because it does not know there is an error in its system model. Therefore, it is not surprising that Sarter and Woods found that pilots mostly found errors through information given in nonautomated displays and instruments (i.e., based on observations between desired and actual aircraft behavior, not on indications of the nominal status of the automated systems). The same phenomenon is true for other types of systems. The problem is complicated by the fact that operators cannot always see what the automation is doing and can only tell by directly observing the reaction of the system or by getting feedback from some independent display. Providing independent feedback and providing more feedback on what the automation is doing can alleviate these problems.

10 Human Factors Safety Analysis

Classic human factors evaluation involves a task analysis and experimentation to better understand the effect of the task design on human performance. Human factors safety analysis is similar, but focuses only on the safety-related aspects of human performance rather than worrying about important, but in this context extraneous, factors such as efficiency, job satisfaction, and so on. For this DFW demonstration project, the cognitive psychologists and human factors experts on our team examined the literature to determine the errors human controllers make in the current ATC system, compared the controllers' current task with their new tasks using FAST, and proposed some hypotheses about the effect of the FAST design on human performance and an experimental paradigm to evaluate those hypotheses.

10.1 Human Error in the Current ATC System

In order to understand potential errors in air traffic control settings enhanced with FAST, we need to understand the errors controllers make under the present system. Then we can look at the changes in operations using FAST and provide some hypotheses about whether past error behavior will decrease, increase, or be unaffected by the new automated tools. Our first step was to review the literature on human error in air traffic control.

This literature is based in part on analysis of aircraft accidents. Unfortunately, this approach is not as straightforward as it sounds. For one thing, aircraft accidents (defined as hull damage or loss of life) are extremely rare events. In addition, accidents occurring in highly redundant systems like aviation often have multiple causes [Die91]. If, for instance, two aircraft are approaching one another at the same altitude, the controller issues an advisory for change of altitude, the pilot reads back the advisory, and the controller listens for the readback as well as continuing to monitor the radar to ascertain whether the change of altitude actually takes place. In addition to this, automatic systems (sometimes in both the cockpit and the TRACON or enroute center) issue warnings if loss of separation is imminent. All of these messages have to be ignored for an accident to occur. And each separate act of oversight may have a separate cause.

It is not surprising that humans are implicated in more than half of all aircraft accidents; they are assigned ultimate responsibility for safety in the system. The proportion of error attributed to humans, also not surprisingly, increases as the equipment becomes more reliable [Nag88]. Nearly 50 percent of all accidents occur during approaches and landings, and 75 percent of these accidents are attributed to human error [Nag88]. This percentage is larger than that of any other phase of flight.

Relatively few errors are attributed directly to the air traffic controller. An analysis by Sears [Sea86] estimated that only 9 percent of the 93 major accidents occurring between 1959 and 1983 were found to be the fault of an air traffic controller. Six percent were the result of communication between ATC and the pilot. In a more recent analysis of accidents, McCoy and Funk [McC91] found that controllers were implicated in 6 of 38 (16 percent) of the accidents occurring between 1985 and 1989.

Another source of information about controller errors lies in reports of incident, also referred to as *operational errors and deviations* (OED). They are more common than accidents, but still fairly infrequent. *Operational errors* occur when a controller allows less than the required minimum separation between aircraft (or between an aircraft and an obstruction). The ARTCC system has tracked incidents automatically since 1984. *Operational deviations* occur when an aircraft enters another controller's airspace without permission and a report is filed by the second controller's facility. Both kinds of information are available through the OED data base at the Office of Aviation Safety. The FAA estimates that 99 percent of all air traffic control errors described in incident reports were due to human errors rather than equipment malfunctions [FAA90].

A third source of information about errors is voluntary reports made by controllers and pilots to the Aviation Safety Reporting System. These are confidential reports of the occurrence of dangerous air traffic situations. They are usually the result of a breakdown in standard procedures or errors. These anonymous reports provide an additional source of information about human error in which a formal violation of separation did not result. Based on this source of information, Morrison and Wright [Mor89] grouped controller errors into two broad categories: communication (clearance composition, readback errors) and control (monitoring and coordination).

10.1.1 Communication

Many believe that communication errors constitute the largest single category of errors in aviation. The Canadian Aviation Safety Board reported that over half of incident reports cite breakdown in communication as a contributing factor [CASB90]. Similar conclusions result from an analysis of errors in enroute control [Rod93]. An early report by Billings [Bil81] suggested that more than 73 percent of the errors resulting in incident reports occur in the transfer of information. The vast major of those—85 percent—occur when the information is transmitted orally.

Communication errors can have several sources. Some errors arise from the imprecision in natural language. The 1977 accident at Tenerife in the Canary Islands was attributed to a confusion about the term "at take off," which was used by the flight crew to mean they were in the process of taking off and interpreted by the controller to mean at the point of taking off. As a result of this confusion, the aircraft in question collided with another aircraft taxiing onto the same runway. A contributing factor was an earlier confusion in which the pilot assumed he was cleared for take off when he received a communication describing the route for which he was cleared *after* take off [Cus94].

An accident that occurred in Cove Neck, New York was attributed to a remark that failed to convey the proper level of urgency. The flight crew notified the controller that they were "running out of fuel" but did not use the word 'emergency.' The controller did not realize the severity of the fuel shortage and failed to clear the aircraft for immediate landing. As a result, the aircraft ran out of fuel and crashed, killed 583 people, one of the worst accidents in aviation history [Cus94].

Problems with the general ambiguity of language can be exacerbated by the speed with which the controller speaks as well as inadequacies in the equipment transmitting the message. Controllers who must relay several messages in a short period tend to speak more rapidly. Sometimes the problem is with the equipment: Interference can cause parts of the transmission to be lost. In addition, the equipment is slow and does not always pick up the beginning of the transmission if the controller starts talking too quickly.

The use of standardized terminology both increases explicitness and helps to overcome technical problems. If a controller wants to deny a taxiing aircraft permission to cross a particular runway because another aircraft is landing on it, the controller should say "Hold short for landing traffic." This utterance has an agreed upon meaning: do not cross the runway. In addition, it has few words in common with a message granting clearance to cross the runway. This uniqueness has a distinct advantage—another phrase such as "remain clear of the runway," under faulty transmission or rapid fire speech, may be misconstrued as clearance. Unfortunately, pilots and controllers sometimes slip back into colloquial English, which naturally defeats the purpose of language standardization.

Confusion in call signs can give rise to a problem inherent in the "party line" nature of controller/aircraft communications. The controller communicates with several aircraft on the same frequency, each identified by a different call sign. The controller, missing the call sign, may misunderstand which pilot is making a request or reading back an advisory or clearance. Likewise, a pilot not hearing the call sign, may mistakenly respond to an advisory or clearance meant for another aircraft. This confusion can result from similar call numbers, pilots or controllers failing to give the call signs, or obscuring of information through faulty transmissions.

Even when the language and transmission are clear and include all of the proper information, the human information processing system may introduce its own brand of interference. People are biased to perceive what they expect to hear. The proper procedure for issuing advisories and clearances is for the controller to speak and the pilot to repeat back what he or she heard so that the controller can check to make sure that the information has been properly received. However, the pilot may expect to receive clearance to a particular altitude because he has requested it or because it is the standard altitude for a particular route. In one incident, a pilot flying at 7,000 feet requested a higher altitude. This request was denied with the remark "Unable, traffic at 8000." The pilot mistook this as clearance to 8,000 and began to climb. Likewise, the controller may perceive what he or she expects to hear, i.e., an accurate readback. In an incident described by Cushing [Cus94], the pilot of an inbound carrier cleared to descend to 12,000 read back 10,000. The mistake was not caught by the controller because the standard altitude for the route was 12,000 and that was what he had issued.

Some misperceptions cannot be explained by expectations, such as when numbers are transposed or altitudes are heard as speeds or headings and vice versa. In addition, some incident reports suggest memory errors. The pilot may not remember all of a long string of advisories. The controller may not remember having issued a clearance or may forget to complete a handoff [Mor89]. Sometimes, distracting situations interfere with memory as well. On one occasion, a flight crew dealing with the consequences of a bird entangled in an engine completely forgot to switch to tower frequency or to request permission to land [Cus94].

Successful communication depends, in part, upon a mental model of the situation that is shared by both the sender and the receiver of the message. When people have different understandings of the current situation, a message may mean different things to each of them. Thus, a potential cause of faulty communication is a discrepancy in the information available to the pilot and controller [SPS95]. A pilot may be unaware of other traffic that causes the controller to issue inconvenient advisories. Because he does not fully understand the situation, the pilot may be reluctant to follow the advisory immediately. Likewise, sometimes the pilot is privy to information not available to the controller. A case in point is the airborne Traffic Alert and Collision Avoidance System (TCAS), which provides a visual display of traffic for pilots as well as warning them of impending separation conflicts. It also provides specific escape maneuvers or *resolution advisories* (for example, an advisory to descend). The resolution advisory, issued directly to the pilot but unknown to the controller, can give rise to unexpected situations on the part of the controller, as when the pilot follows a TCAS advisory to change altitudes. The controller does not expect the aircraft to change course and must then deal with the impact of that unanticipated change on the overall traffic pattern. A similar situation can arise if the pilot sees VFR aircraft[4] unknown to the controller.

10.1.2 Situation Awareness

Information available to the pilot or controller, such as that discussed in the previous section, is part of what has come to be called *situation awareness*. Endsley [ER96] defined situation awareness as "perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future." Many believe that failure in situation awareness is also a prominent cause of air traffic control errors. Evidence for this hypothesis includes the fact that enroute center errors typically take place immediately after the controller returns from break and fails to realize that a critical situation was developing [Red92]. Note that the problem is simply another form safety requirement that we talked about earlier in terms of the computer needing an accurate model of the controlled process (the airspace).

Situation awareness has to do with the accuracy of one's mental model of events and objects in the real world. As such, it is dependent on the amount of attention allocated to the situation (which determines the perceptual elements processed) and memory (which maintains a record of those elements for future problem solving). Both attention and memory have been implicated in controller errors [McC91]. Contrary to reports naming communication as the largest single category of errors (by Rodgers, Billings, and Redding,

[4]VFR aircraft are conducting flight in accordance with visual flight rules, sometimes without transponders.

cited earlier), Stager and Hameluck [Sta89, Sta90] concluded that 60 percent of errors in the air traffic system (based on an analysis of 301 incidents) were caused by insufficient attention, nonrecognition of conflict, and judgment errors.

Redding [Red92] concluded that many of the errors he analyzed could have been avoided by effective monitoring of the radar screen and incoming flight progress strips and by using the information to update situation awareness. Failure in monitoring altitude, heading, and flight path was listed among the causes of controller error identified by Morrison and Wright [Mor89] in an analysis of aviation safety reporting system records. According to Redding, the second greatest source of air traffic control error (after communication) was the misidentification or misuse of radar data (37.6 percent). Controller debriefing reports following an error indicate that controllers themselves attribute inadequate interpretation of radar data as being a causal factor in 33.4 percent of their errors [FAA90].

Redding also concluded that 20 percent of errors in situation awareness result from a mismatch between the controllers expectations and what actually occurred in the airspace. This mismatch becomes a problem when the controller fails to create a backup plan for handling alternative scenario evolutions, or simply fails to update situation awareness to include the unexpected turn of events. An example is when the controller expects an aircraft or another center to take a certain action that it fails to take. In one case, the controller thought that an aircraft had already descended as he advised it to do, even though the radar scope plainly displayed altitude information indicating that it had not [Red92].

10.1.3 Workload

A seemingly likely cause of controller errors is increased workload. Workload in air traffic control is often described in terms of traffic complexity, including such factors as number and type of aircraft, staffing, emergencies, and weather [Rod93]. Air traffic control is renowned for its hectic and stressful environment, and a reasonable hypothesis is that errors increase with workload. Indeed, workload was implicated in a Los Angeles Airport (LAX) accident in 1991 in which a departing commuter plane was placed on the runway in the path of a landing US Air 737 [WMM97]. According to Morrison and Wright [Mor89], heavy workload is the reason cited by many who report safety errors. The fact that many such errors can be traced to a lack of timeliness (not giving a clearance in time or failing to correct a navigational error in a timely fashion) tends to support this idea.

Errors also occur, however, in conditions of low or moderate level workload [Sta91]. Several analysts suggest that increased workload is *not* associated with increased errors. In an analysis of operation irregularities and incidents in Canada, Stager and Hameluck [Sta89] found that more errors occur in periods of low to moderate workload and intermediate volume and complexity. Similar results were reported by the same researchers [Sta90] using a variety of workload measures to analyze 301 operating irregularities in the Canadian air traffic system. They found that 80 percent of all errors occurred in periods of average or below average workload. A Canadian Aviation Safety Board report [CASB90], attributing errors to planning, judgment, and attention lapses, found that such errors take place primarily in periods of *moderate* workload.

Similar results have been found with American enroute controller. In an analysis of OED's occurring between 1985 and 1988, Shroeder and Nye (in [Rod93]) found that 25 percent of errors occurred in conditions of below average complexity and 39 percent occurred in conditions of average complexity. Redding [Red92], in an analysis of enroute center errors found that they occurred predominantly in periods of only moderate complexity—8 aircraft or less. A full 72 percent of errors occurred while controllers were controlling 10 aircraft or less.

Although these findings suggest that it is important to understand controller errors in moderate to low workload situations, because of the lack of base rate information it is not possible to draw conclusions about the relative likelihood of errors under varying workload conditions. Rodgers, however, points out that although the number of aircraft operations handled *increased* over his period of study (1985-1988), the average number of aircraft handled during an OED (operational error and deviation) *decreased*. This result suggests that errors occurring during high workload situations did not increase even though periods of high workload did increase.

Rodgers [Rod93] hypothesized that different types of errors might be associated with high and low workload. A major error was defined as less than half a mile horizontal separation and less than 500 feet vertical separation. He found that, on the one hand, computer entry and flight strip processing errors as well as deficiencies in giving or utilizing relief briefings were slightly more likely with greater numbers of aircraft. On the other hand, errors in coordination between sectors and facilities were slightly more likely under conditions of low complexity. There was no significant relationship, however, between severity of error, as reflected in proximity of the aircraft in a separation violation, and workload.

Rodgers noted that separation errors in his study primarily involved loss of horizontal rather than vertical separation. This result may be due to the fact that horizontal separation is determined by combining information derived from the separation of the two targets on the planview display along with their relative speeds and trajectories. Altitude information, on the other hand, is available directly from the data block. In addition, Rodgers found that most errors occurred when one aircraft was level and the other was either climbing or descending.

10.1.4 Vigilance

Maintaining vigilance in periods of low traffic load is simultaneously boring and demanding. Some maintain that these factors in themselves constitutes an increase in workload (see [WDH96]). Typically, performance of vigilance tasks declines over time; much of this decline is said to take place over the first 30 minutes of a watch [Tei74], As the time spent on vigilance tasks increases, detection rate (i.e., β) is reduced and reaction time increases [DP82]. In air traffic control, vigilance must be maintained to detect loss of separation, altitude deviations, VFR popups, and incorrect pilot readbacks.

Very few studies address vigilance in ATC directly. Simulation studies of vigilance using university students (e.g., [Tha89]) suggest that detection of complex targets was reduced over time as would be expected from the typical vigilance pattern. A study using North American Aerospace Defense (NORAD) operators, however, found that professional controllers in realistic settings may not be as susceptible to the same vigilance decrements [PAOM95].

Thus, it is not clear whether errors are disproportionately associated with vigilance tasks in actual air traffic control settings. Performance of vigilance tasks is particularly important in the context of increased automation, but there is little work addressing this question directly. One likely result of increased automation of controller tasks is that the job will become increasingly one of monitoring operations, i.e., maintaining vigilance.

In summary, the study of human error in the current aviation system shows that a large proportion of errors takes place in the course of information transfer. These errors are due in part to the ambiguity of natural language, confusion about addressee, inconsistency of the equipment, as well as the expectations and understanding of both sender and receiver of the message. Simulation studies of CTAS suggest that the number of overall instructions will be reduced [DEG90], thereby reducing the opportunity for communication errors. At the same time, errors resulting from lack of shared situation awareness may increase in likelihood. We address these issues in the next section.

Another major source of errors in the present air traffic control system is inadequate situation awareness, i.e., the controllers' understanding of the current airspace situation (current state) and its possible evolutions (future states). Again, we feel CTAS is likely to have a direct impact on this aspect of air traffic control. Surprisingly, the research literature suggests that controller errors do not appear to be directly related to workload: A larger proportion of controller errors occur under average or light workload conditions. Although CTAS may serve to lighten the workload, this effect alone cannot be expected to reduce controller errors. Finally, we discovered that there is very little work on the aspect of air traffic control upon which increased automation is likely to have the biggest impact—vigilance.

10.2 Comparative Analysis of Present and FAST augmented ATC procedures

Our step for this project was to review the literature on errors that have been attributed to air traffic controllers. The results are summarized in the previous section. We next compared the current control procedures[5] with those of controllers using FAST. From the combined analyses, we identified potential safety-related human factors problem areas in a FAST-augmented air traffic control system.

We focused our analysis on the duties of the TRACON controller, who handles air-

[5]Much of the analysis of present procedures is based on the work of Mundra [Mun89].

craft in the airspace extending approximately 40 nmi from the airport. Duties are divided usually between controllers who handle departing aircraft and those who handle approaching aircraft, using segregated airspace. Departure and arrival controllers coordinate when potential conflicts arise.

In addition to control responsibilities for the main airport, the TRACON often provides control services for nearby (satellite) airports, which usually have separate control positions. TRACONs that have substantial general aviation traffic also have separate VFR positions. For the purposes of this analysis, we focus on the approach positions for the main Dallas/Ft. Worth Airport (DFW).

Aircraft typically enter the DFW TRACON at three or four feeder gates at the TRACON boundary. The approach controllers duties are further divided between those who control aircraft in the sequencing space after the aircraft have been accepted over the feeder gates (called *feeder controllers*, of which there are usually two) and those who handle aircraft on the final approach pattern, beginning either at the downwind or the base leg of the approach (called *final controllers*, again also usually two). Controllers working the feeder position accept approaching aircraft over feeder fixes, merge them into a well-spaced sequence, and hand them off to the final approach controllers. Final controllers space aircraft for final approach and hand them off to the Tower for landing.

These general duties are identical under FAST. However, the exact procedures used may be quite different, depending on the degree of FAST automation implemented. In the following, we first assume the use of the entire FAST automation capabilities and then note relevant differences between full FAST and passive FAST (a less automated version of FAST). The analysis is divided into stages of the operations: before the aircraft enters the TRACON, handoff from the enroute center, sequencing and spacing, and controlling the aircraft.

10.2.1 Before the Aircraft Enters the TRACON

Under the current system, the TRACON controller receives information about aircraft entering the TRACON airspace from surrounding centers in two forms: flight progress strips and the arrival/departure tabular list.

Flightstrips, printed approximately 30 minutes before the aircraft is anticipated to arrive at the sector, are usually received earlier than aircraft appear on the tab list. The flightstrips, which are part of the interface between the TRACON and enroute computers, provide the TRACON controller with information about the kind and volume of aircraft the TRACON will receive as well as their arrival routes. This information is used by the controller for planning and staffing decisions.

The Arrival/Departure Tabular List appears on the radar screen. For arrival positions, it lists aircraft that will arrive at the TRACON in the next 15 to 20 minutes. An aircraft is dropped from the Tab list once it is picked up by the radar.

The enroute center feeds traffic into the TRACON at a rate specified by the TRACON, usually 5 nmi apart. If traffic flow exceeds the requested rate, the flow is reduced or some traffic is maintained in holding patterns. In addition, center controllers attempt to feed aircraft into the TRACON equally over all the corner posts. Aircraft entering the TRACON at a particular fix are usually routed to the same runway. An uneven flow over the corner posts requires rerouting inside the TRACON to maintain even utilization of all runways. Rerouting within the TRACON is more difficult than changing routes outside.

Under FAST, the TMA (Traffic Management Advisor) generates arrival schedules optimized to provide maximum runway throughput while arrival traffic is still in enroute center airspace. This sequence is displayed on a timeline that can be seen on the TRACON radar scopes before the aircraft enter the sector. Aircraft listed on the timeline are color coded according to the feeder gate by which they will enter. The timeline can be set to display time of arrival at the feeder fix, the FAF (final approach fix) or the runway threshold. It can also be set to display aircraft that are as much as 60 minutes from the specified arrival point. Information for making staffing decisions, therefore, is available much sooner than under the present system. In addition, aircraft approach the TRACON more evenly spaced across feeder gates, requiring less use of holding patterns at the TRACON boundary.

10.2.2 Handoff from the Enroute Center

The main tool of the TRACON controller is a radar screen that depicts the position of all the aircraft in the sector. Each aircraft shown on the screen is accompanied by a *data block* that, under the present system, contains the call sign, altitude, ground speed, and weight class of the aircraft. The symbol of the controller responsible for that aircraft is shown over the center of the actual target, and a status area is designated for the receiving controller's symbol when a handoff is initiated.

The data block can also show a low altitude warning if the aircraft descends below the minimum safe altitude for that terrain and a conflict alert that notifies the controller when legal separation is lost. An additional line on the data block is used for scratch pad information entered by the controller, such as the runway assignment. The type of information entered in the scratch pad is standardized within a particular TRACON. In general, controllers can only see the data blocks of aircraft under their control. However, they are able to see data blocks of other aircraft if they press a 'quick look' key. Typically, arrival controllers set their controls so that they can see all arrival aircraft, regardless of what position is handling them.

When an aircraft is about 10 to 15 nmi from the TRACON, the enroute center computer initiates a handoff to the TRACON computer by transferring information concerning that aircraft. When this event occurs, a flashing light appears on the TRACON feeder controller's scope. The initiating (center) controller's screen shows the receiving (TRACON) controller's symbol in the data tag status block. The receiving controller acknowledges the handoff by positioning the cursor over the icon. The initiating controller's icon flashes a few times to indicate that the handoff has been completed. The initiating controller advises the pilot of the frequency upon which to contact the receiving controller. If anything goes wrong, the center and TRACON controllers can talk by telephone.

The handoff procedure under FAST is identical in the auto-read-accept mode. In addition, FAST provides an autoaccept mode that accepts the aircraft without input from the controller, although the advantages of this feature are unclear. The disadvantage, as discussed below, is that it reduces the controller's role to one of monitoring the handoff.

10.2.3 Sequencing and Spacing

After aircraft have been accepted into the TRACON over the feeder gates, they are assigned by runway by the feeder controller. Aircraft entering a particular feeder gate are usually assigned to the nearest runway. Runway assignments are indicated on the planview display in different ways, depending on the convention of the particular TRACON—New York, for example, uses the position of the leader line (such as straight up or to the left) to indicate the runway assignment while other TRACONs use an entry on the scratch pad.

The feeder controller must then sequence and space the aircraft to feed them safely into the approach pattern where they are handed off to the final controller. Aircraft whose entry point is the farthest away usually enter the approach pattern at the downwind leg while those who are nearer enter it at the base leg.

Aircraft entering the feeder gates are separated mainly by altitude. Arrival traffic from different directions must be merged into final approach streams and eventually brought down to the same altitude. Merging arrival traffic into a desired sequence for the final approach and maintaining required spacing is the primary duty of the approach controller. Thus, a major component of the tasks of both the feeder and final approach controllers is "situation awareness," that is, knowing the relative positions of aircraft in the airspace as well as their probable future positions.

Under the current system, controllers accomplish these tasks using a series of guidelines or heuristics to construct a mental model of the sequence from their understanding if the current positions of all of the aircraft. The controller must first determine the interval required between each pair in the sequence. Aircraft must be at least 1000 feet apart vertically or have horizontal separation of 3 to 5 miles, depending on the weight class of the aircraft (see Appendix A for the detailed requirements). Although altitude can be compared numerically (it appears on the data tag), distance is presumably estimated by mentally converting radarscope distances into nautical miles.

The sequence itself is determined by visualizing a common point the aircraft will cross and estimating the flying time to that point for each of the aircraft. The order in which aircraft arrive at the common point determines the basic sequence. Thus, the sequence is usually obvious or determined by entry into the downwind leg of the pattern. If there is a tie at the sequence point, the controller selects the order based on decreasing overall system delay. Normally, the following guidelines apply:

1. Slower aircraft follow faster aircraft;

2. Heavy jets follow nonheavy aircraft on final approach;

3. An aircraft on final approach precedes an aircraft on the base leg.

Sometimes, after a sequence has been determined the controller must alter it to deal with unexpected situations. VFR aircraft can turn with little or no forewarning and must be merged into the sequence quickly. This task is complex because these aircraft require especially large spacing intervals. In addition, missed approaches must be merged back into the sequence and occasionally there is an emergency that bypasses the sequence altogether.

Although an FAA directive recommends defining the sequence as early as possible, in practice most experienced controllers tend to delay formulating a sequence in their mind. This delay allows them to issue fewer control instructions and to be responsive to differing performance variations of the aircraft as well as to incorporate unexpected results. Clearly, sequencing and spacing aircraft is an extremely computationally intensive cognitive task, and it absorbs much of the controller's mental resources. Not only must the controller solve a complex spatial–temporal problem, but she or he must also maintain a large memory load, including the aircrafts' current positions and characteristics as well as the sequence itself.

Under FAST, the sequence and runway assignments are determined automatically for the controller. These assignments are actually completed by the TMA and submitted to the FAST scheduler even before the aircraft enters the TRACON. As in the present system, the initial runway assignment is based upon the feeder gate by which the aircraft enters the TRACON. The sequencing algorithm constructs a first-come-first-served order based on the aircraft's current route, operating characteristics, and the current weather and runway configuration. The sequence number and runway assignment are shown on the fourth line of the data tag when it first appears on the radar screen.

The sequence of aircraft approaching and currently in the TRACON appears on the timeline on the planview display. Aircraft are ordered on the timeline so that those closest to the outer marker (runway or arrival fix, depending on the timeline setting) are lower on the list. Aircraft move down the list as time advances. The time line shows the STA (scheduled time of arrival) and ETA (estimated time of arrival). The STA is color coded by feeder gate allowing the controller to quickly match the timeline information with aircraft displayed on the radar screen. FAST automatically detects missed approaches or aircraft that have failed to execute a clearance and automatically resequences them. Thus, FAST augmentation greatly reduces both the computational and memory load of the sequencing task freeing the controller to solve other problems. However, the fact that the controller devotes fewer attentional resources to the aircraft and their mutual spatial relationships could reduce situational awareness.

10.2.4 Controlling Aircraft

Once the controller decides upon the sequence, he or she directs aircraft, through a series of instructions, into position and eventually to the final approach path. Although there are no published routes inside the TRACON, there are nominal paths or flow patterns. The degree to which the nominal path is followed is a function of the current traffic level, weather, and runway conditions. Controllers are not required to follow nominal routes at all times, nor can they. In fact, deviation from the nominal routes is one of the tools the controller uses to achieve sequencing and spacing, i.e., vectoring. An example of spacing through vectoring is tromboning on the downwind leg. The controller instructs the aircraft to intercept the downwind leg at a point that achieves the proper interval with respect to other aircraft in the sequence. The path can be lengthened further by delaying the turn to base beyond the end of the standard downwind leg. Controllers report that vectoring judgments are made by envisioning the point in the flight path at which a command to enter the base leg (or final) minimizes conflicts and maintains proper spacing. Occasionally, when traffic is especially heavy, aircraft will be taken off the nominal path altogether and maintained in airborne holding patterns.

Speed control is the other major tool used by the controller to establish a sequence with proper spacing. When traffic is heavy, for instance, controllers tend to slow traffic down to gain more time and get more precise turn to final approach. The controller attempts to maintain spacing within a sequence by keeping changes in airspeed, vector, and altitude at a minimum. Control techniques are to some extent idiosyncratic, however. Some controllers prefer vectoring and some prefer speed instructions. Controllers may also develop preferred headings to issue in the sequencing area. Likewise, they may have preferred width of the base leg for optimal control on the turn to final.

When issuing instructions, the controller must consider several other factors as well. The current wind conditions, for instance, affect how instructions are carried out. The controller sometimes gets a feel for wind conditions by requesting a pilot's airspeed and comparing that with the ground speed shown on the data tag. The controller must also consider variations in piloting: Control instructions can have different outcomes depending on how the pilot controls the aircraft. Controllers come to expect more precise performance from commercial pilots, for example. On the other hand, controllers issue only the most routine instructions to foreign pilots lest they be misunderstood. Occasionally, controllers will choose to issue several small heading changes instead of one big one to reduce their vulnerability to how the pilot will respond. Aircraft performance is another consideration. VFR aircraft must be further behind high performance aircraft when they are in trail because they are more susceptible to turbulence. Likewise, they must be further ahead when they lead because they are slower and more easily overtaken.

The controller juggles these factors simultaneously while making complex spatial temporal judgments and continually evaluating the overall situation o determine if the instructions are being carried out and the desired sequence is emerging. At the same time, controller memory load is high; much of the control plan is maintained in the controller's memory. For standard approaches, specific advisories are issued at standard places. Usually, however, the instructions the controller intends to use deviate from these procedures and must be actively maintained in memory or noted down on the flightstrip. Thus, a large proportion of the memory load has to do with prospective memory, i.e., things you must remember to do in the future. The controller must remember which instructions to issue to which aircraft and where. In addition, she or he must remember to monitor responses to those instructions that have already been issued. Moreover, as soon as the instructions are executed they must be flushed from the controller's memory so they are not confused with future plans. Obviously, in heavy traffic the computational and memory load is extremely high.

Under active FAST, not only are sequencing and runway assignments automated, but the system also monitors aircraft progress and recommends advisories. FAST analyses each aircraft's present position, velocity, altitude, and heading every 15 seconds and updates its ETA, initially assuming the nominal path. FAST compares the DTA to the STA to detect differences, called time errors, which are displayed in orange on the third line of the data tag and on the time line. If the time error is small and the ETA is earlier than the STA, FAST evaluates whether a change in speed will alleviate the error, taking into consideration the aircraft's operating characteristics and the wind conditions. If a reduction in speed solves the problem, the numerical value of the appropriate speed change appears in orange on the fourth line of the data tag. When the aircraft is within 5 nmi of the spot where the speed change advisory should be issued, an orange marker appears in that spot on the planview display. If a speed change will not solve the problem, FAST considers a route change, such as extending the path to the runway: When the aircraft is within 5 nmi of the turn to base (or final approach) that will resolve the time error, the data tag turns blue and a blue turn vector appears on the screen in the location where the instruction should be issued. These same advisories appear in white for aircraft entering the TRACON from the opposite direction.

If the aircraft's ETA is late (compared to the STA), the controller can advise an increase in speed or shorten the aircraft's route. FAST calculates advisories to shorten the route in either the route-intercept mode (RI) or the waypoint-capture mode (WC). In RI mode, FAST searches for an interception of the nominal route by extending the aircraft's current heading to a point on the route that enables the aircraft to maintain its position in the sequence. If the controller selects the aircraft with the mouse, the turn that will accomplish the intercept is shown in yellow. Once the aircraft has intercepted the route, it follows the nominal route to the FAF and FAST provides advisories for the turns to base leg and to final.

In the WC mode, FAST computes a path directly to a waypoint, potentially bypassing much of the standard route. The controller selects the waypoint with the mouse and the waypoint turns yellow. The controller then selects the aircraft and a menu pops up from which the controller selects WC mode. A yellow turn arc is shown that intercepts with a straight line leading to the waypoint. FAST bases its calculations of these advisories on the aircraft's position 10 seconds of flight time into the future to allow the controller time to react. As soon as any advisory is displayed, the ETA on the time line will reflect arrival based on completion of the advisory. If the advisories are not followed, the ETA is readjusted.

Thus, FAST takes on much of both the computational and memory load involved in the task of controlling the aircraft. It calculates the exact amount of time error, posts a notice to the controller, and calculates precise advisories to solve the problem. It also reminds the controller of the instructions when it is time to issue them. The controller must monitor the time line or the data tag to detect the need for heading or speed changes, and the controller must choose the method by which the change will be calculated (route intercept or waypoint capture).

In addition, the controller must then issue the verbal instructions to the pilot and monitor readback. The controller is the ultimate judge as to whether to accept advisory recommendations. However, if the controller chooses to use the FAST advisories, the computationally intensive spatial-temporal calculations and the heavy prospective memory load are almost completely eliminated. In addition, the calculations themselves are much more accurate than the estimating procedures currently used and, as such, reduce the number of adjustments required as well as increase throughput.

10.2.5 Passive FAST

FAST has been implemented using varying levels of automation. Passive FAST provides schedule and runway assignments but does not provide the other advisories described above. The passive FAST sequencing algorithm assumes the nominal route unless the controller enters route change information in the flight plan panel. FAST also takes into account other controller input such as runway assignment changes, missed approaches, and emergency priority for particular aircraft. As with the fully automated version, the sequence number and runway assignment are displayed on the fourth line of the data tag. The landing sequence is displayed on the timeline, which shows ETA and STA and indicates time errors. A bracket on the timeline indicates the earliest and latest possible arrival time of that aircraft based on its engine type, weather, and heading. As in the active version, resequencing is automatic and continuous and takes into account the aircraft's present position and that of all other aircraft. Passive FAST automatically detects and resequences missed approaches and aircraft that have failed to execute clearances.

Although passive FAST does not provide specific advisories, it does provide several types of information to assist in making control decisions. For example, passive FAST assists in detecting future conflicts by providing trend vectors that indicate where the aircraft will be if it continues on its current path.

Similarly, the controller can view the tracking history of a given aircraft that shows the last several fixes of the aircraft as a series of X's. Information about the distance and heading between two objects (VORs, aircraft, airports) can be elicited by dwelling on each object and pressing the space bar. In addition, passive FAST provides a winds panel that shows the winds at six different altitudes above the airport. All this information must be estimated by the controller under the present system. While eliciting the information using FAST probably requires approximately the same level of involvement, the product is much more accurate. For the most part, however, the difference between passive and active FAST appears to be one of degree rather than kind.

10.2.6 Summary of the Comparative Analysis

In general, FAST reduces controller workload [Lee95] and provides more precise advisories that require fewer corrections [SLS95]. Simulations suggest that significantly less airspace is used when controllers employ FAST advisories (due to less off-nominal route vectoring) than when they use the current procedures. With the automation, controllers were found to space aircraft more tightly (without violating minimums) and handled approximately 4.6 more aircraft per hour than without the FAST automation. Controllers involved in the simulations commented that FAST usually confirmed their plan and sometimes came up with a better one, although occasionally controllers missed or ignored advisories [DEG90].

Although these are significant advantages in terms of both increased throughput and decreased workload, they may well result in reduction in attentional involvement in the task. Passive FAST assumes the task of generating landing sequence and runway assignments, although the controller must decide which instructions are best able to accomplish that sequence. Active FAST reduces involvement in both of these tasks. As analyses of previous controller errors suggest, situational awareness, which is a direct result of attention processes, is an area in which the controller is especially vulnerable.

In addition, monitoring tasks are increased with FAST. Controllers monitor the plan-view display for information such as sequence numbers, runway assignments, time errors, and advisories. This shift in duties from those involving active participation to those involving monitoring may increase the necessity for vigilance. At present, the effect of vigilance on air traffic control performance is not well understood. Are reduced situational awareness and increased necessity for vigilance indeed aspects of air traffic control with FAST augmentation? The next section outlines a research program that would address these issues.

10.3 Outstanding Safety Questions

The simulation study by Davis et.al. [DEG90] found that FAST reduces controller workload and results in significantly more efficient use of the airspace. Controllers issued fewer speed and heading clearances, handled more aircraft, and reported a reduction in subjective workload. These results suggest that FAST implementation will greatly improve the efficiency of the air traffic system. In addition, the reduction in workload and instructions issued may well lead to increased safety, to the extent that errors in the present system are attributed to these factors. However, there are potentially negative effects as well. Experts hypothesize that increased automation may decrease situation awareness, increase vigilance requirements, and lead to skill degradation. These issues must be thoroughly investigated to ensure that safety will be maintained.

10.3.1 Decreased Situational Awareness?

Many believe that a reduction in situation awareness is a likely side effect of automation. Situation awareness is the controller's understanding of the current state of the airspace being controlled and the rules that govern its dynamics. It can be thought of as a mental model of the relative positions of aircraft in that space and, as such, it is based on information available on the radar screen.

However, situation awareness is more than that. The mental model also includes knowledge of the dynamics governing the situation that allows the controller to project events and aircraft positions into the future. As such, situation awareness makes use of information stored in long-term memory, such as maps, flight plans, aircraft performance information, procedures, and separation standards [Mor80]. The information about the current state of affairs gleaned from the radar scope combined with general knowledge in long-term memory results in the controller's model of the situation currently active in working memory.

The controller's mental model of the situation serves as a guide for planning, information gathering, and monitoring and, as such, is directly related to performance [Mog91]. As described in Section 10.1, many controller errors have been attributed to a failure in situation awareness. Thus, accuracy of the mental model is essential to controller performance.

The cognitive processes that underlie situation awareness are attention and memory. The more attention one pays to something, the more accurate and memorable is one's mental representation for it. In addition, memory for self-generated information is better than for information presented to one "ready-made" [Sla78]. What this means in terms of air traffic control is that the more thoroughly the controller must process information relating to an aircraft and the greater the percentage of that information that is self-generated, the stronger and more accurately it is represented in the controller's model of the airspace.

Under the current system, for example, the controller must generate the landing sequence by analyzing the relative positions of the aircraft, selecting a point in the approach path that they will all cross, and fast-forwarding the aircraft positions in the mental model until each one crosses that point. This task requires close consideration of each aircraft's operating capacities, speed, altitude, and heading. It is an extremely attention-absorbing task. Both passive and active FAST preclude this entire process by presenting the controller with a completed landing sequence. Clearly providing this information vastly reduces the attentional requirements, and it may reduce the controller's overall understanding and memory for the situation as well. As a result, the controller's model of the airspace may suffer. Note that active FAST takes on even more of the controller's attention-intense tasks by recommending heading and speed adjustments.

Situation awareness may be further reduced by the fact that FAST makes some decisions without consulting the controller, such as automatically accepting handoffs, resequencing aircraft, and making new runway assignments. The results of these actions are available to the controller, but the controller must maintain vigilance in monitoring to detect them.

Extensive research suggests that situation awareness is compromised by reduced attentional requirements (e.g., [WBO80, Hop91, Vor93]. Other research, however, concludes that situation awareness does *not* necessarily suffer from reduced attentional requirements. For example, Vortac et.al. [VEFM] found that automated flightstrips neither reduced attention nor impaired performance. In fact, there was an increase in performance in prospective memory tasks under the automated conditions suggesting that in some circumstances the reduced workload that accompanies automation may serve to *increase* situation awareness.

Automation may impact situation awareness in other ways, however. It may reduce *shared* situation awareness in that FAST-enhanced ATC may substantially alter the controller's model of the situation as compared to that of the pilot. More likely, increased automation may increase the requirements on the controller's understanding of the automation itself. The need for controller awareness of system automation dynamics, processes, inputs, and outputs [KC96] may well *increase* as the automation becomes more complex. A similar increase in the necessity for understanding automation has taken place in aircraft cockpits over the past few years. Several studies have demonstrated that even experienced pilots do not fully understand complex flight management systems [SW92, SW94a, SW94b, SW95a, SW95b]. Moreover, lack of understanding of an automated system has been implicated in a significant number of recent incident reports [VMH95].

The importance of operator understanding of automated systems in the air traffic control environment can be illustrated using an example from FAST. To make an intelligent decision about whether to accept a FAST recommendation, the controller must understand how the solution was generated. The controller would need to know, for instance, whether FAST's recommendation was made in the light of the most recent events, such as a missed approach, a popup, or a delay in heading change. If so, the recommendation can be followed without potential for conflict. However, if the recommendation was formulated before the missed approach took place, the controller needs to consider the joint impact of the current situation and FAST's recommendation on the future state of the airspace. Thus, understanding exactly how FAST generates its recommendation is a vital component of the controller's situation awareness.

In addition, an accurate mental model of the automation is important in detecting and diagnosing problems. FAST calculations, for instance, are dependent to a certain extent on information provided to it by the controller, such as route and runway assignment changes. The automation may provide erroneous recommendations if the controller has made a data-entry error. Detection of such an error might well depend on a thorough understanding of the underlying algorithms by which the system generates its advisories. At the very least, an accurate model of the automation is important for the controller to understand the conditions under which the automation can and cannot be trusted as well as what sorts of errors are likely.

10.3.2 Increased Vigilance Requirements?

Another potential unintended side effect of increased automation is an increase in the vigilance component of the controller's task. The usual findings with respect to vigilance are that there is a significant decrement in response time and target detection rate after about 30 minutes of vigilance monitoring. This problem already exists in cockpit automation. Failures in monitoring of the flight management system and autopilot have been blamed for several aircraft accidents and incidents [Lee92, Mos94, Mui88, Ril94]. This may be due in some cases to complacency, i.e., too much trust in the automation that leads to failure in adequate monitoring [Mos94].

Vigilance requirements in air traffic control increase to the extent that the controllers responsibilities shift from those that involve overt activities to those that involve monitoring. This kind of shift may well take place under FAST. For example, FAST provides the controller with sequence numbers, runway assignments, time errors, and advisories, relieving the controller of the duty to actively generate this information: Using FAST, when the information is required, the controller simply obtains it from the planview display where it is posted.

The designers of FAST have wisely tried to keep the controller involved in all steps of the process. The controller decides whether a speed or heading change is the best solution for a time error, and the controller chooses the method by which a heading correction is calculated (route intercept or waypoint capture). Moreover, the controller is the ultimate judge as to whether to accept advisory recommendations. Finally, the controller issues the verbal instructions to the pilot and monitors readback. Nonetheless, there is a distinct reduction in level of active problem solving required by the controller and a simultaneous increase in the monitoring required. These changes may well increase the vigilance component of ATC, although without systematic investigation it is difficult to tell.

Unfortunately, it is unknown at present how an increase in vigilance requirements would impact the performance of experienced air traffic controllers. Laboratory studies suggest that, after about a half hour of vigilance monitoring, response time increases and target detection decreases [DP82]. However, it is not clear whether these findings are generalizable to the air traffic control settings [PAOM95].

10.3.3 Skills Degradation?

Another potential impact of increased automation is the degradation of skills. If controllers are not required to generate landing sequences and time error solutions, they may become increasingly less adept at performing these tasks. This skill degradation could have two possible negative outcomes. First, lack of practice could serve to erode their understanding of the underlying dynamics of the system [Hop88], which is the long-term memory component of situation awareness. Second, the skills themselves may be slowly eroded with lack of practice [Bai90], compromising the quality of human intervention in the event of an automation failure.

10.4 Experiments to Answer These Questions

During this demonstration of our approach to assuring safety as applied to the DFW TRACON, we did not have time to run experiments to answer the outstanding safety questions we identified in our human factors comparison of current procedures and changes implied by the use of FAST. We have, however, designed experiments that we feel would provide important information about the impact of FAST automation on human ATC information processing and performance in two major areas: situation awareness and vigilance. In addition, these experiments would assess the impact of any decline in performance in the event of a system failure. Although we believe the potential for skill degradation is also an important consideration, we do not believe it can be adequately assessed within the scope of the present project. However, we recommend that skills be assessed periodically and that periodic practice sessions in non-FAST augmented control settings be mandated.

The main questions for which we would seek answers through experimentation are: (1) Does the FAST augmentation alter situation awareness? (2) Does it increase the necessity for vigilance? (3) How do potential changes in situation awareness and vigilance impact performance?

A series of experiments are described to investigate the retrospective memory component of situation awareness, assessing recall memory for information presented in a planview display. Comprehension of future implications are assessed as well. In addition, prospective memory for tasks, such as issuing instructions, are evaluated. A second series of experiments would assess the vigilance component, operationalized as the monitoring of sequence and runway assignment changes and compliance with instructions.

Participants. Participants in all the experiments would be fully trained professional air traffic controllers. Some may participate in more than one phase of the program; they would be paid for their participation. It is important to use professional controllers because of evidence of the limited generalizability of conclusions from experiments using nonprofessionals.

Materials. The most common measurement of situational awareness is the query technique or Situation Awareness Global Assessment technique [End88], in which participants perform control duties in high fidelity simulations, the situation is frozen at random intervals, and the subjects' knowledge is assessed (e.g., [GDOT]). We propose the use of a similar technique in most of the experiments described below.

Procedure. Participants interact with the simulation in one of two main conditions. In the automated condition, the sequence information and runway assignments are provided (Passive FAST). In the non-automated condition, participants generate their own sequences as they do under current procedures. Both groups of subjects perform control duties in a sequence of identical, time-limited scenarios. Scenarios vary systematically by traffic load and complexity, ranging from simple scenarios to those with maximum traffic loads. Presentation order is randomized to prevent subjects from anticipating traffic load. Scenarios are halted periodically, the screen darkened, and the participants' knowledge assessed. Assessment procedures vary by experiment, depending on the particular research question.

Situation Awareness: Retrospective memory component. *Experiment 1:* After participating in the simulation for various periods of time, the simulation is halted and the screen darkened. Participants perform a cued memory test in which they are given a map of the sector, including sector boundaries, navigation aids, and airports. They are asked to indicate the location of each aircraft as well as call sign, altitude, ground speed, heading, sequence number, and runway assignment. Subjects are asked to indicate whether each aircraft was currently engaged in a change of altitude (climbing or descending) or change of heading (right or left turn).
Experiment 2: After participating in the simulation for various periods of time, the simulation is halted and the screen darkened. Participants are asked to judge the proximity of pairs of aircraft, described by their call sign, without the aid of a map.

Situation Awareness: Spatial-temporal projection. *Experiment 3:* After participating in the simulation for various periods of time, the simulation is halted and the screen darkened. Participants are given a map with the present location of all aircraft in the sector and asked to draw the anticipated flight plan of each. Subjects are asked to indicate whether there was any potential loss of separation if all aircraft stayed on course.

Situation Awareness: Prospective memory. *Experiment 4:* After participating in the simulation for various periods of time, the simulation is halted and the screen darkened. Participants are given a map with the present location of each aircraft in the sector and asked to indicate which aircraft had received clearances that had not been completed and whether the aircraft was conforming to clearance when the simulation was halted. They are also asked to indicate which aircraft would need a change of heading or altitude within the next five minutes.

Note: Experiments 1, 3, and 4 may be combined to allow for direct comparison of performance by task types across levels of automation.

Performance in all the above experiments would be assessed by subject matter experts using the FAA's On the Job Training Evaluation Form (OJT: FAA Form 3120-25).

Results and Discussion. Decrease in recall accuracy in either the automated or non-automated conditions would suggest a decrease in situation awareness. In addition, we would assess potential interactions between automation level and traffic load and complexity. One possible prediction is that memory accuracy at the two levels of automation is comparable under light load and decreases in one or both conditions as traffic complexity and load increases.

Similar analyses would be conducted to detect differences in spatial-temporal projection, prospective memory, and interactions with traffic load and complexity. These tasks may well be differentially affected by automation and traffic load. Previous work [VEFM] suggests, for instance, that prospective memory is enhanced under automated conditions.

Job performance (as assessed by subject matter experts using FAA's On the Job Training Form) would be compared across automation conditions to determine whether there are any systematic performance differences between the two groups. Job performance would also be compared with performance on situation awareness measures to detect potential relationships between these variables.

Another method that has been used to assess controller's situation awareness or mental model is multidimensional scaling (MDS). Murphy et.al. [Mur89] suggested that the technique could be used to capture the mental picture of current and future situations and controllers' understanding of ATC rules and operational procedures. Plotting comparison judgments in dimensional stimuli space was used by Lapin [Lap85] to demonstrate that skilled controller's space was 3-dimensional while unskilled controller's was only 2-dimensional.

In experiment 2, participants are asked to make spatial comparison judgments between pairs of aircraft. These would be plotted in dimensional stimuli space and analyzed to detect systematic differences between automated and non-automated groups. Results would be used in conjunction with those yielded in the above analysis to understand better the full impact of automation in situation awareness.

Situation Awareness: Understanding of the automation. *Experiment 5:* Participants are involved in simulated shift change transitions. Scenarios vary in complexity and traffic load. Controllers' ability to detect route change data entry errors by previous controllers is measured over the course of a series of 10 minute scenarios. Participants in this experiment will have some familiarity with FAST, possibly through participation in previous phases of the research program. Detection rate is analyzed as a function of complexity and traffic load.

Vigilance: Monitoring. *Experiment 6:* In this experiment, the impact of automation on vigilance monitoring is assessed in a manner similar to that used by Thackray [Tha89]. Controllers participate in simulations for various periods of time, as above. However, scenarios are designed to systematically vary controller interaction with aircraft (e.g., increased pilot requests). Each scenarios contains several changes in planview display information, such as a change in runway assignment or sequence number, that must be detected through efficient monitoring. Participants interact with the simulation for a period of time, after which the simulation is halted and the screen darkened. Participants are given a recall memory task, as described in Experiment 1 above.

Results and Discussion. These experiments assess awareness of monitored information as a function of interaction level and length of participation. A reasonable prediction is that participants will be less likely to notice and encode in memory changes pertaining to aircraft that require less controller interaction (e.g., increased pilot requests) after periods of participation. This result would suggest a vigilance decrement.

Results from this series of experiments would indicate the impact of automation on situation awareness, vigilance requirements, and performance in air traffic control. From these results, we would be able to assess the impact on system safety as well as the ability of the controllers to maintain safety in the event of a system failure.

11 Operations Research Modeling and Analyses

Introduction

As traffic increases, there is a continuing increase in congestion of arriving aircraft to TRACONs. Various scheduling methods have been discussed to smooth the flow and decrease delays and fuel consumption, however additional complexities increase the burden and risk for the air traffic controller. The question is then, what is the optimum balance between safety and cost efficiency?

The objective of this preliminary research is to illustrate a methodology to evaluate the impact of different scheduling algorithms on delay and fuel consumption, and the consequent trade-offs between efficiency and safety.

For this purpose, traffic destined for a TRACON feeder gate was simulated from the moment when aircraft reached the Center until their arrival at the gate. The overall performance in terms of fuel and delay was analyzed under different CTAS scheduling algorithms. Three of the five algorithms accounted for safety by reducing the number of advisories issued by the controller. These three scheduling algorithms are contrasted with two others that account for minimal fuel burn in the schedule.

Although this preliminary research is limited by the accuracy of the data, the comparisons demonstrate the type of trade-off studies between efficiency and safety that could be performed with this methodology.

The simulation model developed is described in section 11.1. Section 11.2 explains the different scheduling algorithms considered, and section 11.3 shows the simulation details and data. Finally, the results and conclusions are presented in the last two sections.

11.1 Model

In order to set up the model, the Center was divided into four quadrants, each quadrant containing a feeder gate as shown in Figure 34. Each quadrant was assumed to be independent. Three tracks were traced along a quadrant, where each track starts at a point in the Center and finishes at the gate. Therefore, the tracks start at different positions along the quadrant, but all of them finish at the same point. Figure 35 illustrates the three tracks modeled in a quadrant.

A track consists of a set of waypoints equally spaced and different altitude levels. A waypoint is identified by two coordinates, the first one is the distance in units from the starting point and the second one is the altitude, as can be seen in Figure 36. The waypoints in this study are 16.5 nautical miles apart, approximately 2 minutes for an airplane cruising at Mach 0.85. Five altitude levels were used with a vertical separation of 1000 ft. The feeder gate is located at the lowest altitude, FL 330 [TAEO85]. Therefore, all aircraft finish their route at 33000 ft of altitude regardless of the starting altitude. More altitude levels are necessary to more accurately describe the physical system, however this preliminary model was developed with only five levels to demonstrate the potential of this methodology. The model can be extended to include more altitudes and waypoints.

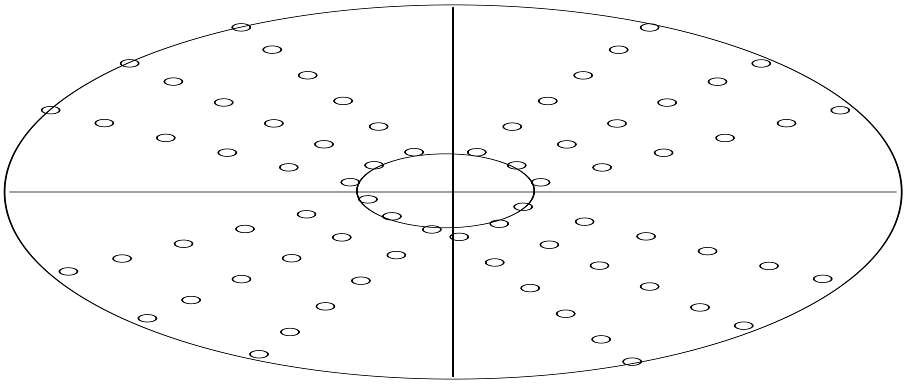


Figure 34: Top View of the Center and TRACON Air Space.

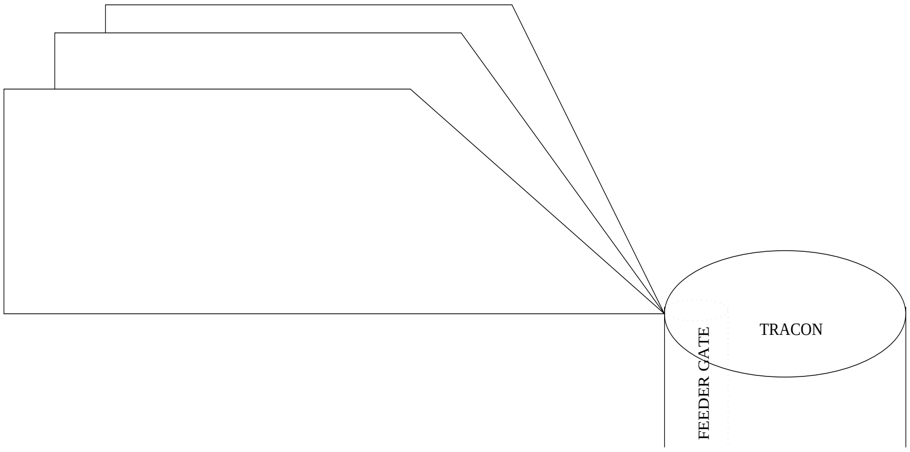


Figure 35: Tracks Along a Quadrant of the Center terminating in the Feeder Gate.

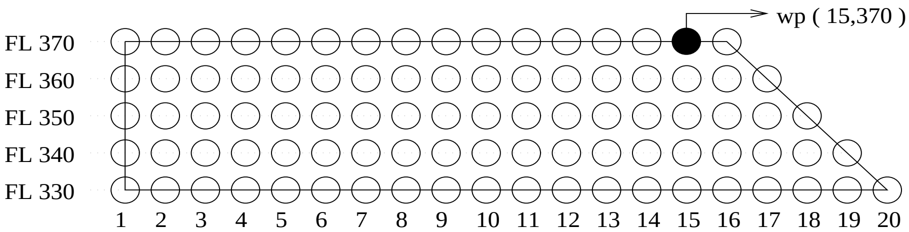


Figure 36: Cross Section of a Track.

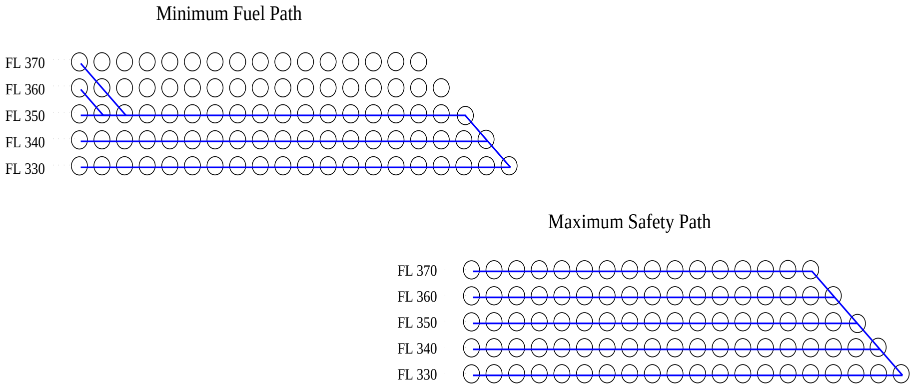


Figure 37: Comparison of Paths. Minimum Fuel Burn vs. Minimum Pilot-Controller Communication.

To maintain the minimum separation required by the FAA, only one airplane is allowed in an arc between waypoints at the same time. Also, aircraft can drop only one altitude level between waypoints. This assumption makes the descent rate approximately 1000 ft. in 2 minutes, or 500 ft./min. on average.

Finally, to establish the trade off between increase in fuel burn and safety, two paths were designed. The first path has minimum fuel burn, and it is based on the assumption that an airplane will always descend to its optimum altitude and maintain it for the whole trajectory until it has to reach the gate. The second path has minimum communications between the pilot and the controller, making it the safest path. In this path, the airplane maintains the starting altitude until the gate, when only one advisory is issued for descent. It is assumed that all airplanes follow either the path with lowest fuel burn or the path with minimum pilot-controller communication. The two paths are shown in Figure 37. In this figure, the path is shown for an aircraft whose optimal altitude is FL 350. It would take two advisories for the airplane to drop from FL 370 to FL 350, hold, and then descend to FL 330. In the safer path, the airplane would hold at its entry altitude (e.g., FL 370) and then descend to FL 330, using only one communication between the pilot and controller.

11.2 Scheduling Algorithms

Five scheduling algorithms were considered in the research. Four of the five were based on a FIFO (First In, First Out) discipline. The current scheduling algorithm implemented in CTAS is based on FIFO. This rule implies that passing is not allowed. Fast aircraft may get stuck behind slower ones, and large gaps in front of slow aircraft may also result.

When an aircraft reaches the Center, its ETA (Estimated Time of Arrival) is calculated. In the basic scheduling algorithm, if an aircraft's ETA is too close to another aircraft's STA (Scheduled Time of Arrival), the controller maintains the FAA minimum separation requirements by delaying the aircraft that arrived last.

Even though a fast airplane is not allowed to pass a slower one at the same altitude, it is

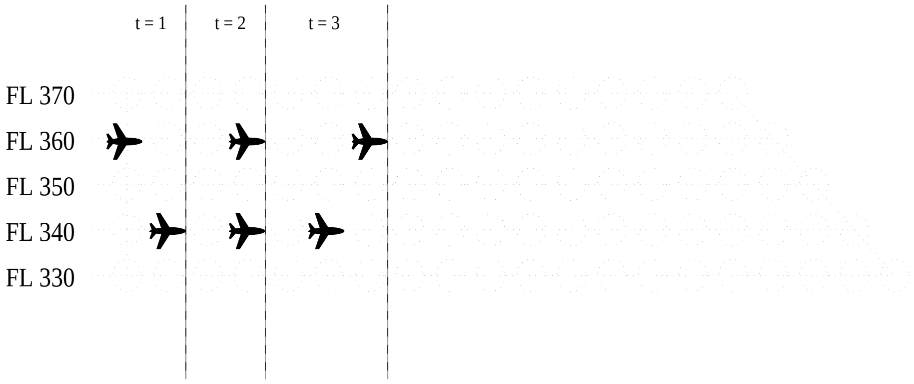


Figure 38: Example of Altitude Overtakes. The Airplane At FL 360 is Faster Than The One At FL 340.

Algorithm	Discipline	Altitude Overtakes	Path
A	FIFO	No	Minimum Fuel
B	FIFO	No	Maximum Safety
C	FIFO	Yes	Minimum Fuel
D	FIFO	Yes	Maximum Safety
E	Allows Horizontal Passing	No	Maximum Safety

Table 2: Summary of Scheduling Algorithms.

possible to have one of the aircraft descend to allow an altitude overtake. For example, two airplanes are shown in Figure 38 where a fast airplane is at FL 360 and a slower airplane is at FL 340. The faster airplane is allowed to overtake the slower one, as illustrated in the figure. Altitude overtakes allow multiple airplanes per geographical location at a time, that is, two aircraft can be at the same waypoint at the same time, as long as they are flying at different altitudes.

Four variations of the basic FIFO algorithm presently used were studied. All scheduling algorithms are summarized in Table 2. In cases A and B, altitude overtakes were not allowed, while cases C and D permitted them. On the other hand, in cases A and C all aircraft followed the path with minimum fuel burn, whereas in cases B and D the safest path was taken.

Finally, algorithm E introduced the possibility to pass horizontally. This algorithm was based on a proposed scheduling algorithm described in [New66]. Algorithm E recognizes two horizons—initial and freeze. The initial horizon was located at the entrance of Center while the freeze horizon was located 165 nm away from the feeder gate.

As aircraft enter the Center and pass the initial horizon, their ETA is calculated and an STA is generated. If a fast airplane arrives and it can pass another aircraft that has not reached freeze horizon yet, then the slower aircraft is delayed to let the faster airplane pass. If an arriving airplane can pass a frozen airplane (already passed the freeze threshold) it may do so only if there is enough space to pass safely without delaying the frozen airplane.

Once an airplane is frozen, it cannot be delayed to allow passing anymore. Examples of this algorithm follow.

Example 1: Current order : A - B - C
Aircraft D enters Center shortly after aircraft C. Both A and B are frozen.
Aircraft D is not fast enough to pass.
$ETA_A < ETA_B < ETA_C < ETA_D$
The algorithm will produce the sequence:
$STA_A < STA_B < STA_C < STA_D$
Aircraft D was delayed to maintain minimum separation

Example 2: Current order : A - B - C
Aircraft D enters Center shortly after aircraft C. Both A and B are frozen.
Aircraft D could arrive between aircraft B and C.
$ETA_A < ETA_B < ETA_D < ETA_C$
The algorithm will produce the sequence:
$STA_A < STA_B < STA_D < STA_C$
Aircraft D was inserted between B and C, aircraft C was delayed once to let D pass and delayed again to maintain minimum separation between D and C

Example 3: Current order : A - B - C
Aircraft D enters Center shortly after aircraft C. Both A and B are frozen.
Aircraft D could arrive between A and B, and there is enough space between A and B to insert D without respacing.
$ETA_A < ETA_D < ETA_B < ETA_C$
The algorithm will produce the sequence:
$STA_A < STA_D < STA_B < STA_C$
Aircraft D was inserted between A and B, no aircraft were delayed

Example 4: Current order : A - B - C
Aircraft D enters Center shortly after aircraft C. Both A and B are frozen.
Aircraft D could arrive between A and B, but there is not enough space between A and B to insert D without respacing
$ETA_A < ETA_D < ETA_B < ETA_C$
The algorithm will produce the sequence:
$STA_A < STA_B < STA_D < STA_C$
Aircraft D was inserted before the first non frozen aircraft, and all aircraft following D were respaced. C was delayed to maintain minimum separation

These algorithms were chosen from a group of possibilities to illustrate a methodology of analysis. In the future, other algorithms or modifications to the ones presented here can be included in the study.

k	F330	F340	F350	F360	F370
1	2.0	1.0	0.0	1.0	2.0
2	1.0	0.5	0.0	0.5	1.0
3	1.0	0.5	0.0	1.0	2.0

Table 3: Fuel Burn Penalty by Flight Level as a Percentage Increase From Optimal Flight Level, F350.

11.3 Simulation Details

The study performed uses a discrete event simulation language (Arena 3.01 [Peg95]) to capture aircraft when they enter the Center (assumed at a distance of 330 nm from the feeder gate at TRACON), and route them to one of three possible tracks. Each airplane follows a path of waypoints where only one aircraft is allowed in an arc between waypoints at a time. If an aircraft arrives at a waypoint and there is another aircraft in that arc, it is sent to a holding pattern represented by a queue where it waits until the arc is free. If an aircraft is going to pass another one horizontally, as in Algorithm E, the delay of the slower aircraft is represented as time in the queue while the passing aircraft occupies the arc.

The simulation was run for 60 aircraft arriving every 2 minutes. The system started empty and finished empty. The track and altitude for each aircraft were randomized following a uniform distribution.

The performance measures were chosen to be delay and fuel burn because of the importance of these factors in the justification of CTAS: "If CTAS were to be implemented nationwide, the airlines would save about a billion dollars per year, mostly from reductions in delay and fuel costs." [Mew97].

The ETA's were calculated for each aircraft by simply obtaining the product of speed and distance for each arc and summing over all arcs on the aircraft's path. The delay was calculated by comparing the initial ETA with the actual arrival time to the feeder gate:

$$Delay = Feeder\ Gate\ Crossing\ Time - ETA$$

The fuel usage was calculated at the end of each arc and it included the fuel consumption during the holding time if an aircraft had to be put on hold. Although fuel flow is a function of several variables, there exists precedent for adopting the assumption that fuel flow remains constant for the time period of interest at the given altitude (see [Tho96]). Deviations from this optimum altitude are modeled as a fuel flow penalty. Table 3 lists the fuel flow penalties used in this study. The equation to determine the amount of fuel burned in a given arc [Tho96] is:

$$Fuel = Average\ Fuel\ Consumption * (1 + Altitude\ Penalty) * Transit\ Time$$

where the average fuel consumption for each aircraft type was taken from is shown in Table 4 [Tho96]. The transit time was calculated as the distance between waypoints (fixed

Mach	Fuel(gal/min)
0.85	60
0.83	48
0.80	26

Table 4: Average Fuel Consumption by Mach Number.

Mach	F330	F340	F350	F360	F370
0.85	494	492	490	490	490
0.83	483	481	479	479	479
0.80	465	464	462	462	462

Table 5: True Air Speed in Nautical Miles Per Hour for Constant Mach Number by Flight Level.

at 16.5 nm) divided by the speed of the aircraft at the particular flying altitude.

Three different types of aircraft were used for the study, where the aircraft were assumed to maintain their listed cruise Mach number during their entire flight (i.e., they do not change speed in level flight, and do not change cruise Mach with altitude) [Tho96]. Due to the lack of real data, the aircraft mix and characteristics were taken from a study of Oceanic Airspace Design [Tho96].

Two cases of speeds modified by altitude were studied. In the first case the speeds are shown in Table 5 [Tho96]. In the second case, the speeds were modified arbitrarily to evaluate the effect of variability, as illustrated in Table 6. However it is noted that these particular speeds may not reflect a real situation.

The capacity at the TRACON feeder gate was varied to study the effect of congestion in the system. Five settings of capacity were implemented:

- 1 aircraft every 0.5 minutes (120 aircraft per hour)
- 1 aircraft every 1 minute (60 aircraft per hour)
- 1 aircraft every 2 minutes (30 aircraft per hour)
- 1 aircraft every 3 minutes (20 aircraft per hour)
- 1 aircraft every 4 minutes (15 aircraft per hour)

Mach	F330	F340	F350	F360	F370
0.85	894	892	890	890	890
0.83	483	481	479	479	479
0.80	165	164	162	162	162

Table 6: Case 2 : Different Speeds to Evaluate Effect of Variability. In Nautical Miles Per Hour.

TRACON	A	B	C	D	E
0.5	0.2	0.2	0.2	0.2	0.2
1	0.3	0.3	0.3	0.3	0.3
2	1.3	1.3	1.3	1.3	1.4
3	28.5	28.5	28.5	28.5	28.6
4	58.0	58.0	58.0	58.0	58.1

Table 7: Average Delay in Minutes for Similar Speeds.

TRACON	A	B	C	D	E
0.5	1887	1893	1887	1893	1890
1	1916	1921	1916	1921	1919
2	2016	2022	2014	2022	2024
3	3272	3278	3272	3278	3279
4	4634	4639	4634	4639	4639

Table 8: Average Fuel Burn in Gallons for Similar Speeds.

Given that the arrivals at Center occur every two minutes, it is reasonable to expect that the system flow is ideal when the capacity at TRACON is greater than 30 aircraft per hour (i.e., less than one airplane every two minutes). Since randomness is introduced, delays when the arrival rate is exactly equal to the leaving rate can be foreseen. Delays and fuel burn are expected to increase when the capacity at the TRACON is smaller than the arrival rate.

11.4 Results

Two sets of runs were completed, the first one corresponds to aircraft at similar speeds and the other one corresponds to aircraft at different speeds.

Tables 7 and 8, and Figures 39 and 40 present the results obtained from the simulation for aircraft flying at similar speeds. Note that the capacity at the TRACON feeder gate severely impacts delay and fuel consumption. As was expected, as soon as the arrival rate exceeds the TRACON capacity, the average delay is enlarged by a factor of 30, while the average fuel burn per aircraft is increased by approximately 1200 gallons.

It is important to point out that for aircraft with similar speeds all the scheduling algorithms have similar performance. When comparing algorithms A and C with B and D (minimum fuel burn vs. minimum pilot-controller communications), the savings in fuel burn are so small that it is better to be safe by generating a minimum number of advisories. On the other hand, when comparing Algorithms A and B against C and D (No altitude overtakes vs. Altitude overtakes), there is not enough difference in delay or fuel consumption that would make the increased risk of allowing altitude overtakes worthwhile.

Algorithm E (Horizontal passing allowed, minimum advisories) is slightly better in terms of delay and fuel consumption than algorithms B and D (minimum advisories) and

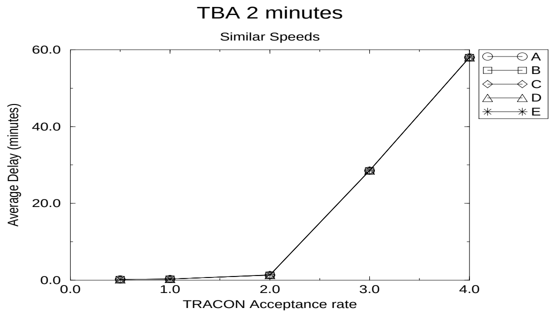


Figure 39: Average Delay for Aircraft with Similar Speeds.

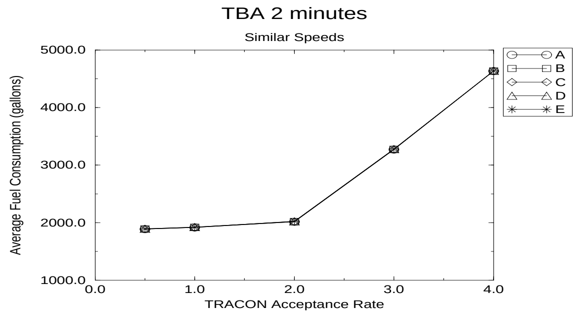


Figure 40: Average Fuel Consumption for Aircraft with Similar Speeds.

slightly worse than algorithms A and C (minimum fuel path). One explanation is that, with all the airplanes flying at roughly the same speeds, the flow is already fairly smooth and there is no opportunity for savings.

Tables 9 and 10, and Figures 41 and 42, present results for aircraft flying at extremely different speeds. When there is a large variation in aircraft speeds, we would expect larger delays and increased fuel consumption, as is illustrated when comparing the tables. In this scenario, Algorithm E demonstrates enormous savings in delay and fuel when TRACON capacity is one airplane every 0.5 minutes.

The effect on delay of having a large disparity in speeds exceeds the effect of TRACON capacity. For the case when all aircraft have similar speeds, the main bottleneck in the system is at the feeder gate to TRACON. When the airplanes have different speeds, intermediate delays are introduced throughout the system by slow aircraft. Scheduling Algorithm E is able to smooth out these intermediate delays by allowing fast aircraft to pass slower ones horizontally.

TRACON	A	B	C	D	E
0.5	38.6	38.6	38.6	38.6	3.4
1	38.9	38.9	38.9	38.9	3.4
2	43.5	43.5	43.5	43.5	4.0
3	59.4	59.4	59.4	59.4	8.3
4	77.9	77.9	77.9	77.9	27.7

Table 9: Average Delay in Minutes for Different Speeds.

TRACON	A	B	C	D	E
0.5	4291	4302	4291	4302	2348
1	4332	4343	4332	4343	2373
2	4587	4598	4587	4598	2447
3	5339	5350	5339	5350	2696
4	6205	6216	6205	6216	3535

Table 10: Average Fuel Burn in Gallons for Different Speeds.

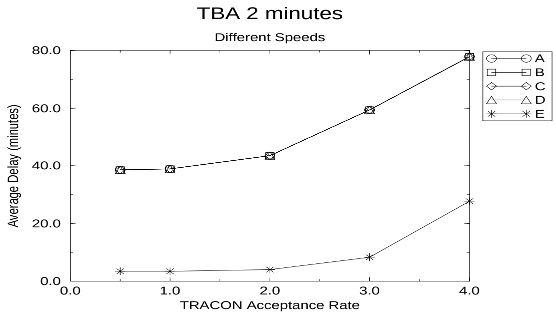


Figure 41: Average Delay for Aircraft with Different Speeds.

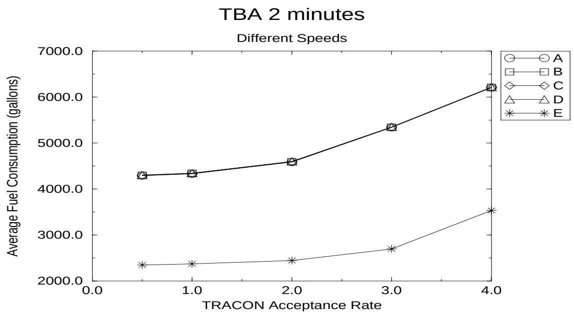


Figure 42: Average Fuel Consumption for Aircraft with Different Speeds

12 Intent Specifications

The types of formal and informal modeling and analysis described in the previous sections together provide a comprehensive assessment methodology. The most effective way to create a safe system, however, is to build safety in from the beginning. The preliminary hazard analysis should start at the earliest concept formation stages of system development and the information should be used to guide the emerging design. Later system and subsystem hazard analysis information is used to evaluate the designs and make tradeoff decisions.

Intent specifications support both general system development and system safety analysis. The design rationale and other information that is normally lost during development are preserved in a single, logically-structured document whose design is based on fundamental principles of human problem solving. Safety-related requirements and design constraints are traced from the highest levels down through system design, component design, and into hardware schematics or software code.

12.1 Goals for Intent Specifications

Intent specifications are based on research in systems theory, cognitive psychology, and human-machine interaction and have several goals.

1. *Bridge between disciplines*: Attempts to build complex systems often run up against the difficulties inherent in communication among the diverse groups needed to complete the project successfully. Specifications should assist in communication and coordination among diverse groups of system designers and builders. One goal of intent specifications is to provide a bridge between the various groups working on a complex system like air traffic control in order to ease coordinated design of components and interfaces and to provide seamless transitions and mappings between the various development and maintenance stages.

2. *Traceability:* A complete safety analysis and methodology for building safety-critical systems requires identifying the system-level requirements and constraints and then tracing them down to the components. After the safety-critical behavior of each component has been identified (including the implications of its behavior when the components interact with each other), verification is required that the components do not violate the identified safety-related behavioral requirements and constraints. In addition, whenever any change is made to the system or when new information is obtained that brings the safety of the design into doubt, revalidation is required to ensure that the change does not degrade safety. This analysis cannot be performed efficiently without the ability to trace design features to specific safety constraints and safety-related design decisions.

3. *Support human problem solving*: Human-centered specifications allow for and support different cognitive and problem-solving styles. They should be based on fundamental

principles of psychology and what is known about human problem solving. A goal for intent specifications is to make it easy for users to extract and focus on the important information for the specific task at hand without limiting the problem-solving strategies used.

4. *Support for ensuring safety and other system qualities*: Essential system-level properties must be built into the design from the beginning; they cannot be added on or simply measured afterward. Up-front planning and changes to the development process are needed to achieve particular objectives. These changes include using notations and techniques for reasoning about particular properties, constructing the system and the software in it to achieve them, and validating (at each step, starting from the very beginning of system development) that the evolving system has the desired qualities. Our specifications must reflect and support this process.

5. *Integrate formal and informal specifications and enhance their interaction*: While mathematical techniques are useful in some parts of the development process and are crucial in developing software for critical systems, informal techniques will always be a large part (if not most) of any complex software development effort. Our models have limits in that the actual system has properties beyond the model, and mathematical methods cannot handle all aspects of system development. To be used widely in industry, our approach to specification must be driven by the need (1) to systematically and realistically balance and integrate informal and mathematical aspects of software development and (2) to make the formal parts of the specification easily readable, understandable, and usable by everyone involved in the development and maintenance process.

6. *Assist in software evolution*: Systems and software are continually changing and evolving; they must be designed to be changeable and the specifications must support evolution without compromising the confidence in the properties that were initially verified. One requirement for correct and safe system evolution is knowing the rationale behind a design. Unfortunately, the reasons why something was done a certain way is often not recorded. Changing a safety-critical system requires determining that the change will not degrade safety. Accidents commonly occur after changes are made to a system. The original designers often think carefully about the safety of their designs and implement effective procedures to eliminate or control hazards. Changes always occur, however, and those who implement them often are not those who originally designed the system. The maintainers do not know why a particular design feature was included and inadvertently remove it while making the change. Recording all the design rationale is difficult and may not be effective if the necessary information is difficult to find when a change is made. Intent specifications record and link the rationale or "why" information directly in the system specification so that it is easy to find and use.

12.2 Intent or "Why" Abstraction

The design of intent specifications is based on the fundamental principles of problem-solving and abstraction that humans use to make complex tasks intellectually manageable. The problems in performing system engineering and software engineering activities such as safety evaluations are rooted in complexity and intellectual manageability. Psychologists have found that complexity itself is not a problem if humans are presented with meaningful information in a coherent, structured context. "People don't mind dealing with complexity if they have some way of controlling or handling it ... if a person is allowed to structure a complex situation according to his perceptual and conceptual needs [New66].

One approach found to be effective in dealing with complexity is hierarchical abstraction, that is, structuring the situation such that the problem solver can transfer the problem to a different level of abstraction. In general systems theory, models of complex systems can be expressed in terms of a hierarchy of levels of organization, each more complex than the one below, where a level is characterized by having *emergent* properties [Che81, Lev95]. The concept of emergence involves the idea that at any given level of complexity, some properties characteristic of that level (emergent at that level) are irreducible. Such properties do not exist at lower levels in the sense that they are meaningless in the language appropriate to those levels. For example, the shape of an apple, although eventually explainable in terms of the cells of the apple, has no meaning at a cellular description level.

Hierarchy theory deals with the fundamental differences and relationships between one level of complexity and another: what generates the levels, what separates them, and what links them. Emergent properties associated with a set of components at one level in a hierarchy are related to constraints upon the degree of freedom of those components. Note that different description languages are required at each level: Describing the emergent properties resulting from the imposition of constraints requires a language at a higher level (a metalevel) *different* than that describing the components themselves.

The goal of hierarchical abstraction is to allow both top-down and bottom-up reasoning about a complex system. In computer science, we have made much use of (1) part-whole (*decomposition*) abstractions where each level of a hierarchy represents an aggregation of the components at a lower level and of (2) information-hiding abstractions where each level contains the same conceptual information but hides some details about the concepts, that is, each level is a *refinement* of the information at a higher level. A higher level of the usual software specifications can be thought of as providing "what" information while the next lower level describes "how." Such hierarchies, however, do not provide information about "why."

Higher-level emergent information about purpose or intent cannot be inferred from what we normally include in such specifications. Design errors may result when we either guess incorrectly about higher-level intent or omit it from our decision-making process. For example, while specifying the system requirements for TCAS using a pseudocode specification, we learned (orally from a reviewer) that crossing maneuvers were avoided in the design for safety reasons. This important safety constraint was not represented in the pseudocode and could not have been unless it had been added in textual comments

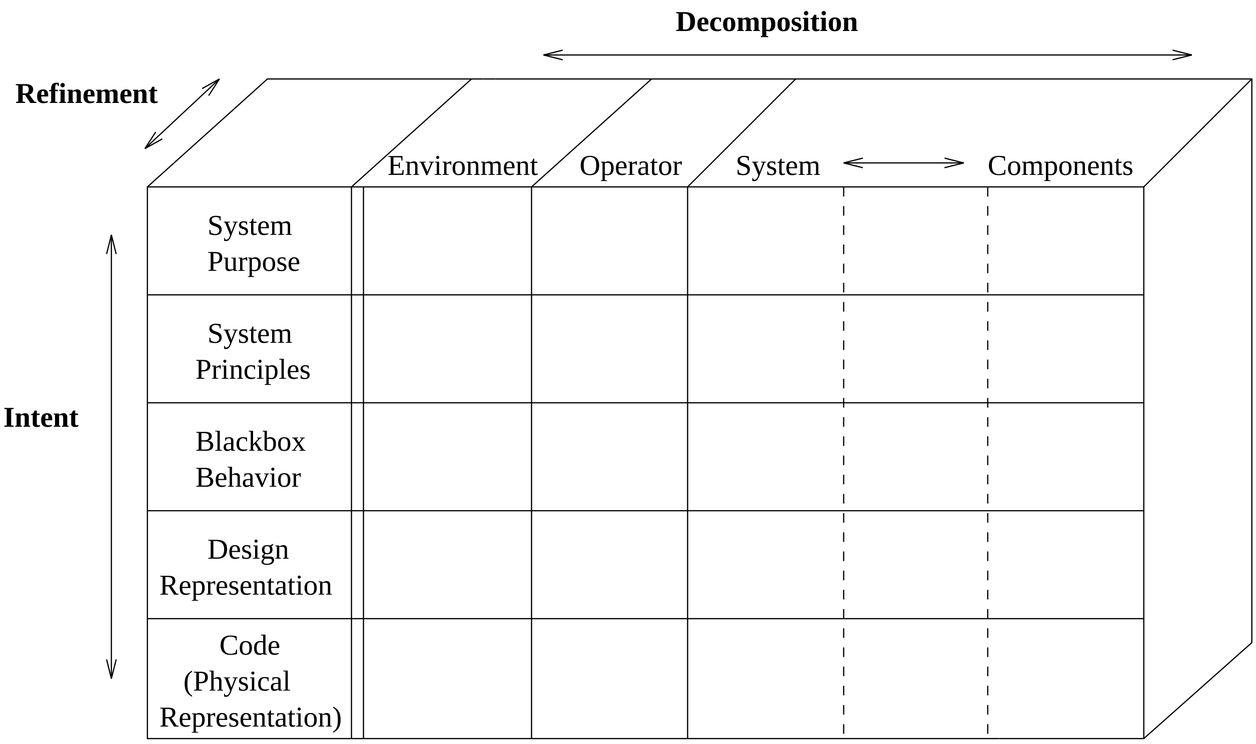


Figure 43: The form of a SpecTRM-RL specification.

somewhere.

Each level of an *intent abstraction* contains the goals or purpose for the level below and describes the system in terms of a different set of attributes or language. Higher level goals are not constructed by integrating information from lower levels; instead each level provides different, emergent information with respect to the lower levels. A change of level represents both a shift in concepts and structure for the representation (and not just a refinement of them) as well as a change in the type of information used to characterize the state of the system at that level.

Mappings between levels are many-to-many: Components of the lower levels can serve several purposes while purposes at a higher level may be realized using several components of the lower-level model. These goal-oriented links between levels can be followed in either direction. Changes at higher levels will propagate downward, i.e., require changes in lower levels while design errors at lower levels can only be explained through upward mappings (that is, in terms of the goals the design is expected to achieve).

12.3 SpecTRM-RL Intent Specifications

System and software specifications in SpecTRM are organized along three abstraction dimensions: intent, refinement, and decomposition (see Figure 43). The vertical dimension specifies the level of intent at which the problem is being considered, i.e., the language or model that is currently being used. The decomposition and refinement dimensions allow users to change their focus of attention to more or less detailed views within each level or model. The information at each level is fully linked to related information at the levels above and below it.

The intent dimension has five levels of abstraction. The highest level of the specification contains the overall goals and safety constraints. Some of the information here is generated through the preliminary hazard analysis process. The next lower level contains the underlying scientific principles upon which the design at the lower levels is based and through which the goals and constraints at the highest level are satisfied. Models and specifications at this level may be subjected to operations research and other types of system analyses to evaluate alternative designs (such as various aircraft spacing and routing alternatives for an ATC system) with respect to the higher-level goals and constraints. The third level contains the black-box functional behavior model of the type we produced for CTAS. This level is the most appropriate one for formal models. The fourth and fifth levels contain design and implementation information. Because each level is mapped to the appropriate parts of the intent levels above and below it, traceability of design rationale and design decisions is provided from high-level system requirements and constraints down to code (or physical form if the function is implemented in hardware) and vice versa.

Each level of an intent specification supports a different type of reasoning about the system and represents a different model of the same system. The model at each level is described in terms of a different set of attributes or language. Level 1 (System Purpose) assists system engineers in their reasoning about system-level properties such as goals, constraints, hazards, priorities, and tradeoffs. The second level allows engineers to reason about the system in terms of the physical principles and laws upon which the design is based. The third level enhances reasoning about the logical design of the system as a whole and the interactions between components as well as the functional state without being distracted by implementation issues. The lowest two levels provide the information necessary to reason about individual component design and implementation issues. The mappings between levels provide the relational information that allows reasoning across the hierarchical levels and tracing from high-level requirements down to implementation and vice versa.

The intent information represents the design rationale upon which the specification is based and thus design rationale is integrated directly into the specification. Each level also contains information about underlying *assumptions* upon which the design and validation is based. Assumptions are especially important in operational safety analyses. When conditions change such that the assumptions are no longer true, then a new safety analysis should be triggered. These assumptions may be included in a safety analysis document (or at least should be), but are not usually traced to the parts of the implementations they affect. Thus the system safety engineer may know that a safety analysis assumption has changed (e.g., the pacemakers are now being used on children rather than the adults for which the design was originally designed and validated), but it is normally a very difficult and resource-intensive process to figure out what parts of the design used that assumption.

Because the separation of human factors and the design of the human-computer interface from the main system and component design can lead to serious deficiencies in each, we have attempted to integrate both types of specifications at each level and across levels. Interface specifications and specifications of important aspects of the system environment are also integrated into the intent specification. Finally, each level of the intent specification includes a specification of the requirements and results of verification and validation activities.

We did not have the information available to complete an intent specification for CTAS or even for FAST. We have created such a specification for TCAS II, however, which is similar to ATC but resides on individual aircraft in order to provide the pilot with collision avoidance information. We took the formal system specification we created for the FAA TCAS project and extended it to provide an example of intent specifications. The table of contents and Level 1 of our specification can be found in Appendix F. The entire specification is too large to include in this report, Readers who are interested can found the example specification at the following URL: www.cs.washington.edu/projects/www/intent/intent.ps

In general, the rationale behind the safety-related design decisions in CTAS needs to be documented. This documentation may exist (we have not been privy to all the documentation that exists for the project) but it is not recorded in the Lincoln Labs system specification for CTAS Build 2. In fact, that document does not include the reasons for any of the design decisions.

13 Conclusions

Although computers seemingly give us the ability to build systems of limitless complexity, spectacular failures such as the FAA's Advanced Automation System and the FBI's CIC demonstrate the difficulty in realizing this goal. The limits seem to be in human ability to intellectually manage the design of such systems—to understand all the potential interactions and behaviors and to assure not only that the goals are achieved but that no adverse or undesired behavior results while achieving the goals.

Engineers have always tried to stretch the limits of the complexity of the systems we are able to build successfully. With the development of digital computers, the intellectual limits have become as important as the technical ones. We are attempting to build systems so complex that humans, without assistance, cannot fully understand them in terms of all the potential interactions and behaviors. We are especially having difficulty building complex systems composed of diverse components, including human, computers, and various types of electromechanical devices. The failure of the FAA in building a new ATC system is a public example, but less well publicized failures abound.

Attempting to replace human designers with "intelligent" computers has not proven to be very successful. Other types of automated tools to assist designers have focused on taking care of housekeeping and routine clerical chores, such as configuration management tools. These tools have been very successful in helping to organize development projects, but they do not help much in augmenting the problem-solving abilities of the developers and in stretching the limits to understanding complex systems.

We believe that stretching those limits will require new methodologies and techniques based on understanding how humans solve problems and providing support for that problem solving process. It will also require new forms of abstraction and visualization that allow designers to understand and analyze systems from different viewpoints. Finally, it will require support for humans trained in different disciplines to work together. Integrated product teams have been helpful to some degree, but the problems in communication and providing common ways to think about systems has limited their success.

SpecTRM is a methodology and tool set to assist in building complex systems made up of diverse components. At its heart is a form of specification called an intent specification designed to support human problem solving. In this report, we have demonstrated how some SpecTRM tools and techniques can be used to assess safety for ATC upgrades and outlined the main components of a comprehensive safety program. Although we were unable to get experienced controllers on our team in the time frame of this project, any real ATC safety program would obviously have to include such expertise. We did try, however, to demonstrate how multiple disciplines might work together by forming such a team for this demonstration.

Because of the nature of the NASA CTAS (and AATT) projects, we concentrated on the early parts of a system's lifecycle. Any real safety program would have to include safety testing and verification activities. It also needs to include a component for the operations phase. During operational use of a system, incident and accident data should be collected and analyzed along with analysis of any changes or modifications. Change analysis uses the same procedures as those used during the original development. Our modular models along with the tracing of safety-related constraints to the design that is part of an intent specification should make it easier to perform this change analysis. In addition, periodic audits should be made to ensure that the assumptions underlying the safety analysis (which are recorded in the intent specification) have not been violated by the natural changes that occur in any system over time.

Assuring safety or any other system-level property for any complex system is difficult and will require sophisticated tools and multidisciplinary teams that span the lifecyle. SpecTRM is an attempt to provide a bridge between disciplines and to stretch the limits of complexity we can handle successfully.

References

[FAA90] Federal Aviation Administration. Profile of operational errors in the national airspace system. Technical report, FAA, 1990.

[Bai90] L. Bainbridge. Development of skill reduction in workload. *Developing Skills with Information Technology*, 1989.

[Bil81] C. Billings. Information transfer problems in teh aviation system. Technical report, NASA, 1975.

[CASB90] Canadian Aviation Safety Board. Report on a special investigation into air traffic control services in canada. Technical Report 90-SO001, Canadian Aviation Safety Board, 1990.

[CO88] J.M. Carroll and J.R. Olson. Mental models in human–computer interaction. in M. Helander (Ed.) *Handbook of Human–Computer Interaction*, Elsevier Science Publishers, pp. 45–65, 1988.

[Che81] P. Checkland. *Systems Thinking, Systems Practice*. John Wiley & Sons, 1981.

[CPWM91] R.I. Cook, S.S. Potter, D.D. Woods, and J.M. McDonald. Evaluating the human engineering of microprocessor-controlled operating room devices. *Journal of Clinical Monitoring*, 7, pp. 217–226, 1991.

[Cus94] S. Cushing. *Fatal words: Communication clashes and airplane crashes*. University of Chicago Press, 1994.

[DP82] D.R. Davis and R. Parasuraman. *The Psychology of Vigilance*. London Academic Press, 1982.

[DEG90] T.J. Davis, H. Erzberger, and S.M. Green. Simulator evaluation of the final approach spacing tool. Technical report, NASA, 1990.

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

[Die91] A.E. Diehl. Human performance and system safety considerations in aviation mishaps. *International Journal of Aviation Psychology*, pages 97–106, 1991.

[End88] M. Endsley. Situation awareness global assessment technique (SAGAT). In *National Aerospace and Electronic Conference*, May 1988.

[ER96] M. Endsley and M. Rodgers. Attention distribution and situation awareness in air traffic control. In *The 40th Annual Meeting of Human Factors Society*, 1996.

[GDOT] S.D. Gronlund, M.R.P. Dougherty, D.D. Ohrt, G.L. Thomson, M.K. Blickley, D. Bain, F. Arnell, and C.A. Manning. Role of memory in air traffic control. in press.

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

[Hop91] V.D. Hopkin. Error models for operating irregularities: Implications for automation. In *Automation and Systems Issues in Air Traffic Control*. 1991.

[Hop88] V.D. Hopkins. Training implications of technical advances in air traffic control. In *Symposium on air traffic control training for tomorrows technology*, pages 6–26, 1988.

[Jaf88] M.S. Jaffe. *Completeness, Robustness, and Safety of Real-Time Requirements Specification*. Ph.D. Dissertation, University of California, Irvine, 1988.

[JL89] M.S. Jaffe, and N.G. Leveson Implications of the man-machine interface for software requirements completeness in real-time, safety-critical software systems. *Proceedings of IFAC/IFIP SAFECOMP 89*, Dec. 1989.

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

[KC96] B.H. Kantowitz and J.L. Campbell. Pilot workload and flight deck automation. In *Automation and Human Performance: Theory and Applications*. 1996.

[Kje87] U. Kjellan. Deviations and the Feedback Control of Accidents. in J. Rasmussen, K. Duncan, and J. Leplat (eds.) *New Technology and Human Error*, pp. 143–156, John Wiley & Sons, New York, 1987.

[LSJM67] D. Landis, C.A. Silver, M. Jones, and S. Messick. Level of proficiency and multidimensional viewpoints about problem similarity. *Journal of Applied Psychology*, 51:216–222, 1967.

[Lap85] Y.A. Lapin. Spacial representation and the activity of the air traffic controller. *Moskovskogo Universiteta Sireya 14 Psikhologiya*, 14:68–70, 1985.

[Lee92] J.D. Lee and N. Moray. Trust, control strategies and allocation of function in human machine systems. *Ergonomics*, 35:1243–120, 1992.

[Lee95] K.K. Lee and T.J. Davis. The development of the final approach spacing tool (FAST): A cooperative controller-engineer design approach. Technical Report 110359, NASA, 1995.

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

[Lev97] N.G. Leveson. Intent Specifications: An Approach to Building Human-Centered Specifications. Submitted for publication.

[LP97] N.G. Leveson and E. Palmer. Designing Automation to Reduce Operator Errors. *Systems, Man, and Cybernetics Conference*, Orlando, October 1997.

[LHHR94] Leveson, N. G., M. Heimdahl, H. Hildreth, and J. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, September 1994.

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

[LS87] N.G. Leveson and J.L. Stolzy. Safety analysis using Petri nets. *IEEE Trans. on Software Engineering*, SE-13(3):386-397), March 1987.

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

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

[McC91] W.E. McCoy and K.H. Funk II. A taxonomy of atc operator errors based on a model of human information processing. In *The Sixth International Symppsium on Aviation Psychology*, 1991.

[Mew97] M. Mewhinney, *Nasa Technology increases efficiency at new airport* (NASA Release 95-66). (http://ccf.arc.nasa.gov/dx/basket/storiesetc/CTAS95-18.html)

[MLRPS97] F. Modugno, N.G. Leveson, J.D. Reese, K. Partridge, and S.D. Sandys. Integrated safety analysis of requirements specifications. *Third IEEE Interational Symposium on Requirements Engineering*, 1997 (also *Requirements Engineering Journal*, 1997).

[Mog91] R.H. Mogford. Error models for operation irregularities: implications for automation. *Automation and Systems Issues in Air Traffic Control*, 1991.

[Mor80] N. Moray. Human information processing and supervisory control. Technical Report AD-A092840, 1980.

[Mor89] R. Morrison and R.H. Wright. Atc control and communications problems: an overview of the recent ASRS data. In *The Fifth International Symposium in Aviation Psychology*, 1989.

[Mos94] K.L. Mosier, L.J. Skitke, and K.J. Korte. Cognitive and social psychology issues in flight crew/automation interation. In *Human Performance in Automated Systems: Current Research and Trends*. 1994.

[Mui88] B.M. Muir. Trust between humans and the design of decision aids. In *Cognitive Engineering in Complex Dynamic Worlds*, pages 71–83. 1988.

[Mun89] A. Mundra. Description of air traffic control in the current terminal airspace environment. Technical Report MTR-88W00167, 1989.

[Mur89] Murphy, Reaux, Stewert, Coleman, and Kelly. Modeling air traffic control performance in highly automated environments. In *Human Factors Society*, pages 47–51, 1989.

[Nag88] D.C. Nagel. Human error in aviation operations. In *Human Factors in Aviation*, pages 263–303. 1988.

[New66] J.R. Newman. Extensions of human capability through information processing and display systems. Technical Report SP-2560, System Development Corporation, 1966.

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

[Peg95] C. Pegden. *Introduction to Simulation using SIMAN*. McGraw Hill, 1995.

[Per84] C. Perrow. *Normal Accidents: Living with High-Risk Technology*. Basic Books, Inc., New York, 1984.

[PAOM95] R.A. Pigueau, R.G. Angus, P. O'Neill, and I. Mack. Vigilance latencies to aircraft detection among NORAD surveillance operators. *Human Factors*, 37(3):624–637, 1995.

[Ras87] J. Rasmussen. Approaches to the control of the effects of human error on chemical plant safety. *International Symposium on Preventing Major Chemical Plant Accidents*, American Institute of Chemical Engineers, February 1987.

[Ras90] J. Rasmussen. Human error and the problem of causality in analysis of accidents. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors in Hazardous Situations*, pages 1–12, Clarendon Press, Oxford, 1990.

[Red92] R.E. Redding. Analysis of operational error and workload in air traffic control. In *Human Factors Society*, 1992.

[Ree96] J. D. Reese and N.G. Leveson Software Deviation Analysis: A Safeware Technique. *AIChe 31st Annual Loss Prevention Symposium*, Houston, March 1997.

[Ree96] J. D. Reese and N.G. Leveson Software Deviation Analysis. *International Conference on Software Engineering*, Boston, Mary 1997.

[Ril94] V. Riley. A theory of operator reliance on automation. In *Human Performance in Automated Systems: Current Research and Trends*, 1994.

[Rod93] M. Rodgers. An examination of operational error database for enroute air traffic control centers. Technical report, FAA Civil Aeromedical Institute, 1993.

[SPS95] E. Salas, D.P. Prince, and L. Shrestha. Situation awareness in team performance: implications for measurement and training. *Human Factors*, 1995.

[SW92] N. Sarter and D.D. Woods. Pilot interaction with cockpit automation: operational experience with the flight management system. *Internation Journal of Aviation Psychology*, 1992.

[SW94a] N. Sarter and D.D. Woods. Decomposing automation: automony, authority, observability and perceived anamacy. In *Human Performance in Automated Systems: Current Research and Trends*, pages 191–197. 1994.

[SW94b] N. Sarter and D.D Woods. Pilot interaction and cockpit automation II. an experimental study in pilots' model and awareness of automation in flight management systems. *International Journal of Aviation Psychology*, 4:1–28, 1994.

[SW95a] N.D. Sarter 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.

[SW95b] N. Sarter and D.D. Woods. Silent, strong, and out of the loop: Properties of advanced (cockpit) automation and their impact on human-automation interaction. Technical Report 95-TR-01, Cognitive Systems Engineering Laboratory, Ohio State University, 1995.

[Sea86] R.L. Sears. A new look at accident contributions and implications of operations and training procedures. Unpublished report from Boeing Commercial Airplane Company.

[Sla78] N.J. Slamecka and P. Graff. The generation effect: delineation of a phenomenon. *Journal of Experimental Psychology: Human Learning and Memory*, 4:592–604, 1978.

[SLS95] R.A. Slattery, K.K. Lee, and B.D. Sandford. Effects of atc automation on precise approaches to closely spaced parallel runways. In *The 1995 A1AA, Guidance, Navigation, and Control Conference*, August 1995.

[Sta91] P. Stager. Error models for operating irregularities: implications for automation. In *Automation and Systems Issues in Air Traffic Control*. 1991.

[Sta90] Stager and Hameluck. Factor in air traffic control operating irregularities. Technical Report TP 9324E, Transport Canada, 1990.

[Sta89] Stager, Hameluck, and Jubis. Underlying factors in air traffic control incidents. In *The 33rd Annual Meeting of Human Factors Society*, 1989.

[Tei74] W.H. Teichner. The detection of a simple visual signal as a funtion of time on watch. *Human Factors*, 16:339–353, 1974.

[Tha89] R.I. Thackray and R.M. Touchstone. Detection efficiency on air traffic control monitoring task with and without computer aiding. *Aviation, Space and Environmental Medicine*, 60:744–748, 1989.

[Tho96] S. Thorarinsson. An optimization approach to oceanic airspace design. Master's thesis, University of Washington, 1996.

[TAEO85] L. Tobias, L. Alcabin, H. Erzberger, and P.O'Brien. Simulation studies of time control procedures for the advanced air traffic control system. Technical Report 2493, NASA, 1985.

[Tuf90] E.R. Tufte. *Envisioning Information*. Graphics Press, 1990.

[VMH95] S.F. Vakil, A.L. Midkiff, and R.J. Hansman. *Mode awareness problems in advanced autoflight systems*. MIT Aeronautical Systems Lab, 1995.

[Vor93] O.U. Vortac. *Should Hal open the pod bay doors? An argument of modular automation*. Embry Riddle Aeronautical University Press, 1993.

[VEFM] O.U. Vortac, M.B. Edwards, D.K. Fuller, and C.A. Manning. Automation and cognition in air traffic control: An emperical investigation. *Applied Cognitive Psychology*.

[WDH96] J.S. Warm, W.M. Denber, and P.A. Hancock. Vigilance and workload in automated systems. In *Automation and Human Performance: Theory and Applications*. 1996.

[WBO80] D. Whitefield, R.D. Ball, and G. Ord. Some human factors aspects of computer aiding concepts for air traffic controller. *Human Factors*, 22:569–580, 1980.

[WMM97] C. Wickens, A.S. Mavor, and J.P. McGee. *Flight to the Future: Human Factors in Air Traffic Control*. National Academy Press, 1997.

A Minimum Separation Standards

Separation Standards

Source: Thompson, S.D., *Terminal Areas Separation Standards: Historical Development, Current Standards, and Processes for Change*, 16 January 1997

RADAR SEPARATION

In en route airspace

Below FL600	5 miles
At or above FL600	10 miles
Both aircraft are within 40 miles of the radar antenna and below FL180	3 miles

In terminal area

Both targets are less than 40 miles from the antenna	3 miles
Either target is 40 miles or more from the antenna site	5 miles

VERTICAL SEPARATION

Below FL290	1,000 feet
At FL290 and above	2,000 feet

WAKE VORTEX SEPARATION

General Wake Vortex Separation Standards	
Heavy behind Heavy	4 miles
Large or Heavy behind a Boeing 757	4 miles
Small or Large behind a Heavy	5 miles
Small behind a Boeing 757	5 miles

Landing Wake Vortex Separation Standards	
Small behind a Large	4 miles
Small behind a Boeing 757	5 miles
Small behind a Heavy	6 miles

TERMINAL RADAR SEPARATION REQUIREMENTS FOR APPROACH

Separation may be reduced to 2.5 miles on final (within 10 miles of the runway threshold) under the following conditions:

The leading aircraft's weight class is the same or less than the trailing aircraft

The leading aircraft is not a Heavy or a Boeing 757

The average runway occupancy time is a documented 50 seconds or less

Radar displays are operational in the tower

Runway turnoff points are visible from the control tower

VERTICAL SEPARATION DURING ILS OPERATIONS TO PARALLEL RUNWAYS

Vertical separation during vectoring to approach in the case of parallel runway ILS operations is maintained by having the aircraft using one runway maintain an altitude at least 1,000 feet higher than the aircraft using the other runway.

DEPENDENT PARALLEL ILS OPERATIONS

Diagonal separation requirements when conducting operations to two parallel runways in IMC	
Runway centerlines are between 2,500 and 4,300 feet apart	1.5 nautical miles
Runway centerlines are between 4,300 and 9,000 feet apart	2 nautical miles

B Partial Fault Tree

This appendix contains a partial ATC fault tree. For the most part, only the parts of ATC related to CTAS and passive FAST (e.g., arrival traffic in the TRACON area) are considered. Because of the size of the fault tree, we did not finish it but only included enough for the reader to get a feel for what such a fault tree might contain.

The standard fault tree notation, while very easy to read, is not space efficient. The size of our fault tree made a horizontal format more practical than the normal vertical format.

A pair of controlled aircraft violate minimum separation.

OR

- Violation of minimum in-trail separation while aircraft on final approach to same runway.

  OR

  - A/C are in violation of in-trail separation standards when initially assuming in-trail relationship (merging incorrectly)

    OR

    - Conflict between in-trail a/c and turbo-prop traffic that was split off earlier but is now being vectored to remerge with the jet traffic.
    - Aircraft merged incorrectly with traffic on opposite downwind
    - Aircraft on downwind fails to execute a turn-to-base in timely manner and resequenced incorrectly.
  - A/C initially established in valid trail but later violate separation standards

    OR

    - Excess positive closure rate exists upon initial establishment of in trail relationship and a/c do not maneuver sufficiently to avoid violation [a/c will compress to less than legal separation if continue with published procedure.]

      [schedule generated by TMA contains conflicts]
    - Change in closure rates causes violation of in trail separation standards => A
  - Unscheduled arrival (popup, missed approach, emergency) leads to in-trail separation violation
- Violation of distance or time separation between streams of aircraft landing on different runways.

  OR

  - Two aircraft on final approach to parallel runways not spatially staggered.
  - Two aircraft landing consecutively on different runways in intersecting or converging operations violate minimum difference in threshold crossing time.
  - An aircraft violates the non-trangression zone while airport is conducting simultaneous independent ILS approaches to parallel runways.
  - Aircraft fails to make turn from base to final approach
- Violation of minimum separation between arrival traffic and departure traffic from nearby feeder airports or overflight traffic.

A Change in closure rates causes violation of in trail separation standards

AND
- One or more aircraft change ground speed
  - OR
    - Weather disrupts aircraft trajectory (e.g., tailwinds)
    - Pilot changes speed
      - OR
        - TCAS advisory says to change speed
        - Pilot misprograms autopilot
        - Pilot mishears speed advisory
    - Autopilot error
- Maneuver required to avoid violation of in trail separation standards
- Insufficient or no corrective maneuvering occurs
  - OR
    - Insufficient speed change in leading or trailing aircraft
      - AND
        - Pilot does not recognize problem on his/her own
          - AND
            - Does not recognize visually (by see and avoid)
            - TCAS does not provide alert
        - Controller instructions do not cause aircraft to make necessary speed change
          - OR
            - Controller does not issue speed advisory => A.1
            - Controller issues speed advisory but pilot does not receive it. => A.2
            - Controller issues speed advisory, pilot receives but does not follow it. => A.3
            - Controller issues speed advisory that does not avoid separation violation. => A.4
            - Controller issues speed advisory but too late to avoid separation violation. => A.5
            - Controller issues speed advisory to wrong aircraft. => A.6
    - Insufficient path stretching occurs => B

A.1 Controller does not issue speed advisory.

OR
- Controller does not perceive the conflict.
  - AND
    - Conflict alert does not cause controller to perceive advisory.
      - OR
        - Controller believes conflict alert is false alarm.
        - No conflict alert is displayed.=> A.1.1
        - Controller does not see conflict alert
          - Alert removed from screen before sees it.
    - Controller does not preceive conflict based on displayed information (poition, route, speed data)
      - OR
        - Controller misinterprets displayed information
          - Flight data blocks in misleading position
        - A/c data unavailable
          - OR
            - Not displayed at all
              - Plane not identified on plan view display (no flight data block displayed)
                - OR
                  - Incorrect handoff
                  - Loss of radar tracking data
                  - Display failure
            - Data displayed but not long enough to perceive it.
        - Displayed a/c data incorrect with respect to actual situation (airspace state)
          - OR
            - Recent sensor data missing for one or both a/c (a/c displayed in coast)
            - Automation provides wrong a/c information
              - OR
                - Automation software flaw
                - ta corrupted but not detected by automation
        - Controller distracted or inattentive
- Controller preceives conflict but too late to avoid separation violation
  - OR
    - Controller distracted
    - Display delayed
- Automation failure and backup procedures do not work
  - OR
    - Controllers out of practice
    - Equipment not maintained

A.1.1 No conflict alert displayed

OR
- Controller has suppressed display of conflict alert

  OR
  - For one or both a/c individually
  - For region in which one or both a/c currently displayed
  - For his or her entire display
- Automation fails to declare conflict

  OR
  - Too much recent sensor data missing for one or both a/c

    OR
    - Hard sensor failure
    - Sensor coverage failure
      - Permanent shadow zone
      - Transient failure (atmospherics)
    - Communication failure
      - Data not received
      - Data received but discarded as corrupted
  - Automation decision not to declare conflict is incorrect.

    OR
    - Software conflict detection logic flaw
    - Software corrupted
    - Data corrupted (undetected by automation)
  - Automation conflict detection not being executed

    OR
    - Task not dispatched properly by operating system
    - Computer hardware failure.
    - Fast not computing trajectories (so unable to detect a conflict)
      - Controller has dirrected FAST to assign a given runway.

A.2 Controller issues appropriate speed advisory but pilot does not receive it.

OR
- Physical communication failure
  - OR
    - Radio failure
    - Radio on wrong frequency
- Human communication failure
- Controller issues advisory to wrong aircraft
  - OR
    - Psychological slip
    - Wrong label associated with aircraft on planview display
    - Right label on aircraft but in misleading place on screen

A.3 Controller issues appropriate speed advisory, pilot receives it, but does not follow it.

OR
- Pilot distracted by other alert or emergency.
- Pilot does not believe based on own evaluation and believes will endanger safety of aircraft
- TCAS issues conflicting advisory.
- Aircraft automation causes advisory not to be followed.
  - OR
    - Pilot enters wrong information into FMS
    - Information entered right, automation error
    - Advisory incompatible with aircraft automation (doesn’t react the way controller assumes it will).

A.4 Controller issues speed advisory that does not avoid separation violation.

OR
- Controller error
  - Doesn't use CTAS speed advisory and miscalculates required speed.
- Automation error
  - CTAS uses inaccurate models of aircraft dynamics and propulsion system performance
  - CTAS uses incorrect atmospheric data.
  - CTAS uses incorrect algorithm.

A.5 Controller issues speed advisory but too late to avoid separation violation.

OR
- CTAS does not provide information in timely manner.
- Controller distracted by other problem.
- Advisory needed at handoff point and handoff delayed
  (2 controllers set up to autoaccept and wrong one assigned aircraft)

B. Insufficient path stretching occurs.

AND
- Pilot does not recognize problem on his own
- Controller instructions do not cause aircraft to make necessary path changes

OR
- Controller does not issue appropriate advisories
  - OR
    - Automation failure and backup does not work
      - OR
        - Controllers out of practice
        - Equiptment not maintained
        - Increased traffic load due to automation and controllers unable to handle with manual methods.
    - Controller does not know advisory needed
      - OR
        - Mental model flawed (incorrect view of situation)
        - Overconfidence in automation and automation error occurs
      - STA to metering gate is outside aircraft's accessible time range for speed control but PS not displayed in data block for aircraft
  - Controller issues incorrect advisory.
    - OR
      - Autopath stretching tool provides wrong information
        - Miscalculates position along the off-course direction in front of the aircraft where required delay will have been absorbed.
        - Miscalculates required delay.
          - Incorrect algorithm
          - Does not update location of path stretch marker in response to changing conditions.
        - Magnetic heading displayed to capture waypoint is incorrect.
      - Controller misreads path stretching information
      - Controller does not use path stretching tool and incorrectly calculates required heading, speed, and latitude advisories
      - Controller does not issue a new heading vector that returns a/c to predesignated capture waypoint on original route when a/c reaches marker.
        - Distracted, etc.
- Controller issues appropriate advisories but pilot does not receive them.
- Controller issues, pilots receives but does not follow
- Controller issues advisories but too late to avoid separation violation

C Sample ATC Requirements and Constraints

This section should be taken as an example only of safety-related requirements and design constraints for ATC upgrades. We made no attempt to be complete.

C.1 General ATC Safety Requirements

1. ATC shall provide advisories that maintain safe separation between aircraft.
2. The system shall protect against pilots not following advisories, aircraft automation errors, and any other event that causes an aircraft to deviate from an approved path or trajectory while under ATC control.
   (a) ATC shall warn aircraft that enter an unsafe atmospheric region.
   (b) ATC shall provide timely warnings to aircraft to prevent their incursion into restricted airspace.
   (c) ATC shall provide conflict alerts and shall do so early enough that a violation of minimum separation standards (i.e, a near midair collision) can be avoided.
   (d) ATC shall provide alerts and advisories to avoid intruders if at all possible.
3. ATC shall provide weather advisories and alerts to flight crews.
4. ATC shall provide advisories that maintain safe separation between aircraft and terrain or physical obstacles.
5. All IFR aircraft within the ATC system shall be under positive control (there is two-way radio communication with a controller who has responsibility for that aircraft) until the pilot and ATC agree to terminate IFR services.
6. The ATC system shall be compatible with TCAS and other aircraft navigation systems.
7. The system shall be able to operate safely in the event of the failure of any component(s).
   (a) There must be backup or emergency (contingency) procedures in case of radar or computer failure.
   (b) Backup or emergency procedures must be capable of maintaining adequate safety in the worst case airspace situation including that made possible (for example, increased traffic or decreased separation) by introducing automated aids.
   (c) At a minimum, controllers must have access to TBD (flight strip information) in the event of computer failure.

General ATC Design Constraints

1. ATC must not issue advisories to an approach aircraft that would cause a conflict with overflight traffic.

2. ATC must not cause or contribute to an aircraft executing a controlled maneuver into terrain.

3. ATC must not issue advisories that direct aircraft into areas with unsafe atmospheric conditions.

4. ATC must not issue advisories that direct aircraft into restricted airspace unless avoiding a greater hazard.

5. ATC must not issue advisories outside the safe performance envelope of the aircraft.

6. ATC advisories must not distract or disrupt the crew from maintaining safety of flight.

   - ATC must not adversely affect workload in the cockpit so as to degrade safety of flight.

7. ATC must not issue advisories that the pilot or aircraft cannot fly or that degrade the continued safe flight of the aircraft.

8. ATC must not issue advisories that cause an aircraft to fall below the standard glidepath or intersect it at the wrong place.

9. ATC must not provide an advisory to an aircraft that thwarts a TCAS maneuver and leads to an NMAC (near midair collision) unless the goal is to protect against a greater hazard.

   Note: TCAS has related and complementary design constraints: (1) it must not interfere with the ground-based ATC system and (2) it must generate advisories that require as little deviation as possible from ATC clearances.

C.2 Automated System Requirements and Constraints

The specific requirements and constraints on the automation will depend on what the automation has been tasked to do. The following are some example constraints that might apply to the current CTAS automation or to some of its planned upgrades.

The most general or high-level constraints on the automation might be:

1. The automated system must not interfere or reduce the ability of the controller to maintain safety of the airspace.

Note: As long as we hold the pilot responsible for the safety of the aircraft and the controller for the safety of the airspace, then we must hold ATC system designers responsible for not interfering with the pilot's or controller's exercise of their responsibility.

2. The automated system must not create hazards that would not have existed had the automation not been installed.

In addition to these general constraints, more specific requirements and constraints might be specified such as:

1. CTAS shall provide conflict alerts early enough that the conflicts can be avoided.
2. CTAS shall minimize maneuvering advisories for areas of planned close separation (such as approaches to parallel runways).
3. CTAS shall maintain a consistent global ordering (sequence numbers) for each runway.
4. The schedule generated by the TMA must be conflict free to within specified separation requirements at the runway and at the metering fix.
5. Speed advisories and turn advisories generated by FAST must always maintain safe separation.
   (a) If the trajectory and conflict resolution logic (1) detects that a jet currently in-trail with other jets will compress to less than legal separation if it continues on the published STAR procedure or (2) detects a future conflict with a merging turbo-prop on downwind or in merging with traffic on the opposite downwind, then FAST must provide a speed advisory that slows the jet in order to avoid loss of legal separation at the merge point. FAST must not fail to provide the speed advisory, provide it too late, or provide an incorrect advisory.
   (b) FAST must not provide an incorrect turn advisory to the final approach course, fail to display it to the controller at the point at which to issue the turn, provide it at the wrong point, fail to provide one when needed or provide it too late.
6. The automated system shall be able to detect unscheduled arrivals such as missed approaches and popups, aircraft that have failed to execute a clearance, emergencies, disturbances, etc. and display a new set of advisories in time to avoid a hazard or to mitigate any hazards if it is not possible to avoid the hazard entirely.
7. After the freeze point, CTAS must not change runway assignments except to avoid a hazard.

8. New controller aids must not interfere with the safety-related functions of any existing ATC components that the new aid is not replacing or changing.

9. The automated system must not negatively affect controller workload in a way that could lead to an increase in safety-related human errors.

   Note: This design constraint may involve either increases or reductions in workload (causing reduced monitoring ability) and leads to design decisions such as FAST giving highest priority to pulling a dissimilar engine or weight class out of the traffic stream when doing runway allocation and reallocation.

   Note: An alternative way of stating this might be that CTAS must not increase controller workload to the point where human error is increased nor decrease it to the point where monitoring ability is reduced.

10. New controller aids must not degrade the functions provided by the existing ATC system.

11. If human controllers and pilots are assigned monitoring responsibility for automated subsystems, then they must be provided with independent means to detect errors in the information provided by or erroneous behavior of those subsystems.

12. Automation upgrades must be compatible with TCAS and other aircraft systems. Automated aids must operate safely for all possible combinations of aircraft types and on-board automation.

13. The automated system must not generate advisories that will confuse the pilot or exceed the ability of a human to follow or use them or distract the pilot during critical phases of flight or when handling emergencies.

    (a) FAST must avoid making runway changes late in the traffic flow (i.e., near the feeder gates) in order to avoid undesirable increases in workload for pilots of arriving aircraft (in terms of late changes in selecting navigation frequencies and configuring the aircraft for an approach).

14. The automated system must never produce information (trajectories, advisories, sequences, or runway assignments) that will lead to a hazard.

    (a) FAST must never assign two aircraft to the same runway or in any other way produce runway assignments that will lead to a hazard.

       Assumption: Because the controllers cannot see on their screens all the information that FAST has, runway assignments are likely to generate trust in controllers and they are very likely not to question them. Therefore, the controllers should not be relied upon to provide monitoring and error detection in computer-generated runway assignments.

(b) CTAS must not generate speed, altitude, or turn advisories that lead to a violation of minimum separation standards.

(c) CTAS must not create an advisory for an arrival aircraft that would cause a conflict with overflight traffic or lead the aircraft into restricted space unless avoiding a greater hazard was the goal.

(d) CTAS must not create advisories that cause an aircraft to fall below the standard glidepath or to intersect it at the wrong point.

(e) CTAS must not generate trajectories and/or advisories outside the safe performance envelope for the aircraft involved, such as advisories that significantly reduce stall margins or result in stall warnings or that require the aircraft to maintain high vertical rates of climb or descent, except to prevent a greater hazard.

(f) Path stretching advisories must not lead to a hazard.

15. The upgrades must not reduce situational awareness of the controllers or the flight crews in any way that could decrease safety (lead to a hazard).

    (a) The system must provide controllers with the information needed to maintain an adequate mental model of the traffic situation and present it effectively (e.g., in a way that controllers will notice it and can use it without unreasonable effort).

    (b) Controller vigilance must not be reduced by the automated aids.

    (c) Pilots must be aware of and able to understand and evaluate any trajectories that might be given either directly to the pilot or through a data link.

16. The system design must reduce the number and complexity of the controls to a minimum consistent with safe system operation.

17. The computer controls used by ATC controllers in all possible combinations and sequences must not result in a condition or state of the automation whose presence or continuation could lead to a system hazard.

18. Displays must not fail to provide information needed by the flight crew or the controller to avoid hazards and must not confuse the flight crew or controller as to the current state of the airspace system.

    (a) If an advisory or alert is generated by the automation, it must not be removed before the operator has seen it and it is no longer applicable or needed by the controller to safely control traffic.

    (b) Information displayed by the automated system must not be delayed beyond the time that the controller can use it appropriately.

(c) High-priority (safety-related) alerts must be emphasized to the controller, must be easily readable, and must never be hidden or overlapped by other screen displays.

(d) Changes to safety-related displays (e.g., runway assignments or sequence numbers) must be denoted in such a way that the controller (or pilot) knows that a change has been made (e.g., CTAS must in some way highlight the fact that a runway assignment has changed and perhaps require an acknowledgment if workload tradeoffs are determined to be acceptable)[6].

> **Note:** The controller can manually override the runway assignment. In the absence of an override, FAST will assume that the controller has accepted the assignment it has generated.

(e) High-priority (safety-related) alerts must never be removed without acknowledgment of the controller unless it has become obsolete. If alert queues are used, obsolete information should, in most cases, be marked and allowed to scroll off the screen rather than simply being eliminated.

(f) High-priority (safety-related) alerts must never be eliminated to reduce screen clutter. Screen clutter must be minimized according to a precedence based on safety.

(g) If there is a need to remove information from the screen before the information becomes obsolete, a priority should be established to eliminate it in order of least criticality and potential contribution to a hazard.

(h) Graphical advisories should be used where possible.

(i) The potential for confusing information about one aircraft with that for another must be eliminated or reduced as much as possible.

19. The automation must never cause the controller to send an advisory to the wrong aircraft.

    (a) CTAS must never put the wrong label on an aircraft.

    (b) CTAS must not place information on the screen for one aircraft such that it can be confused with another aircraft.

[6] For aircraft on final approach, FAST issues speed advisories only if necessary to maintain safe separation. FAST documentation states that in order to reduce display clutter in the final approach zone, FAST advisories are displayed "only briefly but long enough for controllers to observe them." How is it determined that the controller has observed them or how long this period should be?

D Completeness Criteria for Black-Box Requirements Analysis

A requirements specification describes the required black-box behavior of the component. Although design information is sometimes included in requirements specifications, the safety analysis described in this report is concerned only with black-box behavior, which is the only aspect of a component specification that can directly affect system hazards.

The requirements specification defines the function to be implemented. The goal of completeness analysis basically is to ensure that the model of the process used by the automated controller—in our case, described using a state machine model—is sufficiently complete and accurate that hazardous process states do not occur. We have defined completeness criteria for each of the state machine parts: the states, the transition (triggering) events, the inputs and outputs, and the relationship between the transition events and their related outputs.

The following list contains only the criteria without discussion or description. For more information about the criteria, see *Safeware* [Lev95].

D.1 General Considerations

Completeness requires that both the characteristics of the outputs and the assumptions about their triggering events be specified:

$$\mathit{trigger} \Longrightarrow \mathit{output}$$

In response to a single occurrence of the given stimulus or trigger, the program must produce only a single output set. A black-box statement of behavior allows statements and observations to be made only in terms of outputs and the externally observable conditions or events that stimulate or trigger them (the *triggers* for short). In terms of the state machine, this restriction means that both the states and the events on the transitions must be externally observable.

Not only must the output be produced given a particular trigger, but it must *not* be produced without the trigger:

$$\mathit{trigger} \Longleftarrow \mathit{output}$$

where output may be a single output or a set of outputs (such as periodically repeated output) all triggered by a single event.

A complete trigger specification must include all conditions that trigger the output, that is, the set of conditions that can be inferred from the existence of an output. Such conditions represent assumptions about the environment in which the program or system is to execute. For black box requirements, the terms of the trigger condition must involve only observable phenomena external to the program whose behavior is being specified, that is phenomena visible at the blackbox boundary (inputs and outputs, the passage of time, sense switches).

D.2 State Completeness

The operational states will, of course, be specific to the system. But in general, these states can be separated into normal and non-normal processing modes, and completeness criteria can be applied to the transitions between these modes.

Transitions from normal operation to non-normal operation are often associated with accidents. In particular, when computers are involved, many accidents and failures stem from incompleteness in the way the software deals with startup and with transitions between normal processing and various types of partial or total shutdown.

- *The system and software must start in a safe state. Interlocks should be initialized or checked to be operational at system startup, including startup after temporarily overriding interlocks.*
- *The internal software model of the process must be updated to reflect the actual process state at initial startup and after temporary shutdown.*
- *All system and local variables must be properly initialized upon startup, including clocks.*
- *The behavior of the software with respect to inputs received before startup, after shutdown, or when the computer is temporarily disconnected from the process (off-line) must be specified, or it must be determined that this information can be safely ignored, and this conclusion must be documented.*
- *The maximum time the computer waits before the first input must be specified.*
- *Paths from fail-safe (partial or total shutdown) states must be specified. The time in a safe but reduced-function state should be minimized.*
- *Interlock failures should result in the halting of hazardous functions.*
- *There must be a response specified for the arrival of an input in any state, including indeterminate states.*

D.3 Input and Output Variable Completeness

The inputs and outputs represent the information the sensors can provide to the software (the controlled variables) and the commands that the software can provide to the actuators (to change the manipulated variables). These input and output variables and commands must be rigorously defined in the documentation.

At the black-box boundary, *only* time and value are observable by the software. Therefore, the triggers and outputs must be defined only as constants or as the value and time of observable events or conditions. Events include program inputs, prior program outputs, program startup, and hardware-dependent events such as power-out-of-tolerance interrupts. Conditions may be expressed in terms of the value of hardware-dependent attributes accessible by the software such as time-of-day clocks or sense switches.

- *All information from the sensors should be used somewhere in the specification.*
- *Legal output values that are never produced should be checked for potential specification incompleteness.*

D.4 Trigger Event Completeness

The behavior of the control subsystem is defined with respect to assumptions about the behavior of the other parts of the system—the conditions in the other parts of the control loop or in the environment in which the controller operates. A *robust* system will detect and respond appropriately to violations of these assumptions (such as unexpected inputs). By definition, then, the robustness of the software built from the specification depends upon the completeness of the specification of the environmental assumptions—there should be no observable events that leave the program's behavior indeterminate. These events can be observed by the software only in terms of trigger events, and thus completeness of the environmental assumptions is related to the completeness of the specification of the trigger events and the response of the computer to any potential inputs.

D.4.1 Robustness Criteria

- *To be robust, the events that trigger state changes must satisfy the following:*
  1. *Every state must have a behavior (transition) defined for every possible input.*
  2. *The logical OR of the conditions on every transition out of any state must form a tautology.*
  3. *Every state must have a software behavior (transition) defined in case there is no input for a given period of time (a timeout).*

D.4.2 Nondeterminism

- *The behavior of the state machine should be deterministic (only one possible transition out of a state is applicable at any time).*

D.4.3 Value and Timing Assumptions

Ensuring that the triggers in the requirements specification satisfy the previous four criteria is necessary, but it is not sufficient for trigger event completeness. The criteria ensure that there is always exactly one transition that can be taken out of every state, but they do not guarantee that all assumptions about the environment have been specified or that there is a defined response for all possible input conditions the environment can produce. Completeness depends upon the *amount* and *type* of information (restrictions and assumptions such as legal range) that is included in the triggers. The more assumptions about the triggers included, the more likely that the four above criteria will ensure that the requirements include responses to unplanned events.

In real-time systems, the times of inputs and outputs are as important as the values. Therefore, both value and time are required in the characterization of the environmental assumptions (triggers) and in the outputs.

Essential Value Assumptions

Value assumptions state the values or range of values of the trigger variables and events. An input may not require a specification of its possible values. A hardwired hardware interrupt, for example, has no value, but it may still trigger an output. When the value of an input is used to determine the value or time of an output, the acceptable characteristics of the input must be specified, such as range of acceptable values, set of acceptable values, or parity of acceptable values.

- *All incoming values should be checked and a response specified in the event of an out-of-range or unexpected value.*

Essential Timing Assumptions

The need for and importance of specifying timing assumptions in the software requirements stem from the nature and importance of timing in process control, where timing problems are a common cause of runtime failures. Two different timing assumptions are essential in the requirements specification of triggers: timing intervals and capacity or load.[7]

Timing Intervals. While the specification of the value of an event is usual but optional, a timing specification is *always* required: The mere existence of an observable event (with no timing specification) in and of itself is never sufficient—at the least, inputs must be required to arrive after program startup (or to be handled as described in other criteria).

- *All inputs must be fully bounded in time, and the proper behavior specified in case the limits are violated or an expected input does not arrive.*
- *A trigger involving the nonexistence of an input must be fully bounded in time.*

[7] *Load* here refers to a rate, whereas *capacity* refers to the ability to handle that rate.

Capacity or Load. In an interrupt-driven system, the count of unmasked input interrupts received over a given period partitions the computer state space into at least two states: normal and overloaded. The required response to an input will differ in the two states, so both cases must be specified.

- *A minimum and maximum load assumption must be specified for all interrupt-signaled events whose arrival rate is not dominated (limited) by another type of event.*

- *A minimum-arrival-rate check by the software should be required for each physically distinct communication path. Software should have the capacity to query its environment with respect to inactivity over a given communication path.*

- *The response to excessive inputs (violations of load assumptions) must be specified.*

The requirements for dealing with overload generally fall into one of five classes:

1. Requirements to generate warning messages.
2. Requirements to generate outputs to reduce the load (messages to external systems to "slow down").
3. Requirements to lock out interrupt signals for the overloaded channels.
4. Requirements to produce outputs (up to some higher load limit) that have reduced accuracy or response time requirements or some other characteristic that will allow the CPU to continue to cope with the higher load.
5. Requirements to reduce the functionality of the software or, in extreme cases, to shut down the computer or the process.

The first three classes are handled in an obvious way. The behavior in the fourth and fifth classes (commonly called performance degradation and function shedding) should be graceful—that is, predictable and not abrupt.

- *If the desired response to an overload condition is performance degradation, the specified degradation should be graceful and operators should be informed.*

- *If function shedding or reconfiguration is used, a hysteresis delay and other checks must be included in the conditions required to return to normal processing load.*

- *Operators must be informed about the conditions leading to graceful degradation and the actions taken as a result of these conditions.*

D.5 Output Specification Completeness

As with trigger events, the complete specification of the behavior of an output event requires both its value and its time.

- *Safety-critical outputs should be checked for reasonableness and for hazardous values and timing.*

D.5.1 Environmental Capacity Considerations

The rate at which the sensors produce data and send it to the computer is the concern in input capacity. Output capacity, on the other hand, defines the rate at which the actuators or operators can accept and react to data produced by the computer. If the sensors can generate inputs at a faster rate than the output environment can "absorb" or process outputs, an output overload might occur.

- *For the largest interval in which both input and output loads are assumed and specified, the absorption rate of the output environment must equal or exceed the input arrival rate.*

Output load limitations may be required because of physical limitations in the actuators (such as a limit on the number of adjustments a valve can make per second), constraints on process behavior (excessive wear on actuators might increase maintenance costs), or safety considerations (such as a restriction on how often a catalyst can be safely added to a chemical process).

Differences in input and output capacity result in the need to handle three cases:

1. The input and output rates are both within limits, and the "normal" response can be generated.
2. The input rate is within limits, but the output rate will be exceeded if a normally timed output is produced, in which case some sort of special action is required.
3. The input rate is excessive, in which case some abnormal response is necessary (graceful degradation).

- *Contingency action must be specified when the output absorption rate limit will be exceeded.*

- *Update timing requirements or other solutions to potential overload problems, such as operator event queues, need to be specified.*

- *Automatic update and deletion requirements for information in the human–computer interface must be specified.*

Events placed in operator interface queues may be negated by subsequent events. The requirements specification should include the conditions under which such entries may be automatically updated or deleted from a queue. Some entries should be deleted only upon explicit operator request; however, workload may be such that the entries must be queued until the operator can acknowledge them. For example, when an air traffic control operator asks for the count of aircraft whose velocity exceeds a certain speed, the response may be queued and should not disappear until the operator acknowledges receipt.

Some queued events may become irrelevant to the operator, such as information about a warning to an air traffic controller that an aircraft is too close to the ground or to ground-based hazards such as tall antennas (called a minimum safe altitude warning or MSAW).

The warning itself may be shown on the situation display, but additional information that cannot be displayed may be put into a queue. If the portion of the queue that contains the MSAW-related information is not currently visible to the operator, it may be removed from the queue automatically when the MSAW is removed from the situation display. If that portion of the queue *is* currently visible, the queued information should not be removed: Operators generally find it distressing when information disappears while they are looking at it or while they are temporarily glancing away.

There could be safety implications as well. Suppose that there are MSAWs for two separate aircraft, but the queue display can accommodate only one event at a time. The operator might glance back at the display, not realizing that the first event has been removed and replaced by the second. The operator would then read the recommended course for the second aircraft and transmit it to the first aircraft, not realizing that the event data he or she is reading is not the same data seen a second or two before.

- *The required disposition for obsolete queue events must include specification of what to do when the event is currently being displayed and when it is not.*

In general, obsolete event data currently being displayed cannot be automatically deleted or replaced. It may be modified to show obsolescence and removed when the operator indicates to do so or when the overall display is modified in such a way that the obsolete event display becomes invisible (for example, the queue is advanced and the obsolete information is scrolled off the display).

D.5.2 Data Age

Another important aspect of the specification of output timing involves data obsolescence. In practical terms, few, if any, input values are valid forever. Even if nothing else happens and the entire program is idle, the mere passage of time renders much data of dubious validity eventually. Although the computer is idle, the real world in which the computer is embedded (the process the computer is controlling) is unlikely to be. Control decisions must be based on data from the current state of the system, not on obsolete information.

- *All inputs used in specifying output events must be properly limited in the time they can be used (data age). Output commands that may not be able to be executed immediately must be limited in the time they are valid.*

- *Incomplete hazardous action sequences (transactions) should have a finite time specified after which the software should be required to cancel the sequence automatically and inform the operator.*

- *Revocation of a partially completed action sequence may require (1) specification of multiple times and conditions under which varying automatic cancelation or postponement actions are taken without operator confirmation and (2) specification of operator warnings to be issued in the event of such revocation.*

D.5.3 Latency

Since a computer is not arbitrarily fast, there is a time interval during which the receipt of new information cannot change an output even though it arrives prior to the actual output action. The duration of this latency interval is influenced by both the hardware and the software design. An executive or operating system that permits the use of interrupts to signal data arrival may have a shorter latency interval than one that uses periodic polling, but underlying hardware constraints prevent the latency from being eliminated completely. Thus, the latency interval can be made quite small, but it can never be reduced to zero. The acceptable length of the latency interval is determined by the process that the software is controlling.

The choice of operating system, interrupt logic, scheduling priority, and system design parameters will be influenced by the latency requirements. Also, behavioral analysis of the requirements to determine consistency with process functional requirements and constraints may not be correct unless the value of this behavioral parameter is known and specified for the software. Therefore, the requirements specification must include the allowable latency factor.

- *A latency factor must be included when an output is triggered by an interval of time without a specified input and the upper bound on the interval is not a simple, observable event.*

- *Contingency action may need to be specified to handle events that occur within the latency period.*

- *A latency factor must be specified for changeable human–computer interface data displays used for critical decision making. Appropriate contingency action must be specified for data affecting the display that arrives within the latency period.*

- *A hysteresis delay action must be specified for human–computer interface data to allow time for meaningful human interpretation. Requirements may also be needed that state what to do if data should have been changed during the hysteresis period.*

D.6 Output to Trigger Event Relationships

Some criteria for analyzing requirements specifications relate not to input or output specifications alone but to the relationship between them. Although, in general, the relationship depends on the control function being specified, basic process control concepts can be used to generate criteria that apply to all process control systems, such as feedback and stability requirements.

Responsiveness and spontaneity deal with the actual behavior of the controlled process and how it reacts (or does not react) to output produced by the controller. In particular, does a given output cause the process to change, and if so, is that change detectable by means of some input? Basic process control models include feedback to provide information about expected responses to changes in the manipulated variables and information about state changes caused by disturbances in the process.

- *Basic feedback loops, as defined by the process control function, must be included in the software requirements. That is, there should be an input that the software can use to detect the effect of any output on the process. The requirements must include appropriate checks on these inputs in order to detect internal or external failures or errors.*

- *Every output to which a detectable input is expected must have associated with it: (1) a requirement to handle the normal response and (2) requirements to handle a response that is missing, too late, too early, or has an unexpected value.*

- *Spontaneous receipt of a nonspontaneous input must be detected and responded to as an abnormal condition.*

D.7 Specification of Transitions between States

Requirements analysis may involve examining not only the triggers and outputs associated with each state and the relationship between them, but also the paths between states. These paths are uniquely defined by the sequence of trigger events along the path. Transitions between modes are particularly hazardous and susceptible to incomplete specification, and they should be carefully checked.

D.7.1 Reachability

- *All specified states must be reachable from the initial state.*

Most state-based models include techniques for reachability analysis. In complex systems, complete reachability analysis is often impractical, but it may be possible in some cases to devise algorithms that reduce the necessary state space search by focusing on a few properties. The backward-reachability hazard analysis techniques for state machine models outlined in Section 7 are examples of algorithms that limit the amount of the reachability graph that must be generated to get enough information to eliminate hazardous states from the requirements specification.

D.7.2 Recurrent Behavior

Most process control software is cyclic—it is not designed to terminate under normal operation. Its purpose is to control and monitor a physical environment; the nature of the application usually calls for it to repeat one single task continuously, to alternate between a finite set of distinct tasks, or to repeat a sequence of tasks while in a given mode. Most systems, however, include some states with noncyclic behavior such as temporary or permanent shutdown states or those where the software changes to a different operating mode.

- *Desired recurrent behavior must be part of at least one cycle. Required sequences of events must be implemented in and limited by the specified transitions.*

- *States should not inhibit the production of later required outputs.*

An *inhibiting state* for an output is a state from which the output cannot be generated. If every state from which the output can be generated is unreachable from an inhibiting state, then the output cannot be generated again once the inhibiting state is reached. Whether or not this condition represents an incompleteness depends upon the application.

D.7.3 Reversibility

In a process control system, a command issued to an actuator often can be canceled or reversed by some other command or combination of commands. This capability is referred to as *reversibility*.

- *Output commands should usually be reversible.*

- *If x is to be reversible by y, there must be a path between the state where x is issued and a state where y is issued.*

D.7.4 Preemption

When the same physical resource, such as a data entry device or display, must be used in distinct multistep actions at the human–computer interface, requirements will be needed to deal with preemption logic. In addition, some actions may have to be prohibited until others are completed.

- *Preemption requirements must be specified for any multistep transactions in conjunction with all other possible control activations.*

In general, there are three possible system responses to an operator action from a parallel-entry source prior to completion of a transaction initiated by some previous control activation: (1) normal processing in parallel with the uncompleted transaction, (2) refusal to accept the new action, and (3) preemption of the partially completed transaction.

If preemption is possible, then the attempted activation of a multistep sequence requiring the use of a resource already involved in another incomplete transaction provides the following three choices:

1. The new request could completely cancel the previous, incomplete transaction, clearing or replacing any displays associated with it.
2. The new request could preempt the shared resources, but the displayed state could be preserved and restored upon completion of the new transaction.
3. The operator could be prompted and required to indicate the disposition of the incomplete transaction, in which case there will in general be four alternatives:

a. Cancel the incomplete transaction and start the newly requested one.
b. Complete the old transaction and then proceed automatically with the new request.
c. Cancel the new request and continue with the old, incomplete transaction.
d. Defer but do not cancel the old, incomplete transaction.

If any transactions are deferred and restored, obsolete information must be identified, as discussed previously.

D.7.5 Path Robustness

For most safety-critical, process-control software, there are concerns beyond pure reachability: Even if a state fulfills all reachability requirements, there is still the question of the *robustness* of the path, or paths, affecting a particular state.

Suppose that every possible path from a state where command X is issued to any state where command Y is issued includes the arrival of input I. Then if the computer's ability to receive I is ever lost, perhaps due to sensor failure, there are circumstances under which it will not be possible to issue a Y command. The loss of the ability to receive I is said to be a *soft failure mode* because it *could* inhibit the software from providing an output with the value Y.

If the receipt of I occurs in every path expression from *all* states that produce X commands to *all* states that produce Y commands, then loss of the ability to receive I is now said to be a *hard failure mode* because it *will* inhibit the software from producing a Y command.

- *Soft and hard failure modes should be eliminated for all hazard-reducing outputs. Hazard-increasing outputs should have both soft and hard failure modes.*

The more failure modes the requirements state machine specification has, whether soft or hard, the less robust with respect to external disturbances will be the software that is correctly built to that specification. Robustness, in this case, may conflict with safety. A fail-safe system should have no soft failure modes, much less hard ones, on paths between dangerous states and safe states. At the same time, hard failure modes are desirable on the paths from safe to hazardous (but unavoidable) states. An unsafe state, where a hazardous output can be produced, should have at least one, and possibly several, hard failure modes for the production of the output command.

- *Multiple paths should be provided for state changes that maintain or enhance safety. Multiple inputs or triggers should be required for paths from safe to hazardous states.*

In general, operators should be provided with multiple logical ways to issue the commands needed to maintain the safety of the system so that a single hardware failure cannot prevent the operator from taking action to avoid a hazard. On the other hand, multiple interlocks and checks should be associated with potentially hazardous human actions—such as a requirement for two independent inputs or triggers before a potentially hazardous command is executed by the computer.

D.8 Constraint Analysis

In addition to satisfying general completeness criteria, the requirements must also be shown to include the identified, system-specific requirements and to be consistent with the identified system design constraints.

- *Transitions must satisfy software system safety requirements and constraints.*
- *Reachable hazardous states should be eliminated or, if that is not possible (they are needed to achieve the goals of the system), their frequency and duration reduced.*

It is not always possible to enforce a requirement that the software cannot reach hazardous states—sometimes a hazardous state is unavoidable. But this possibility should be known so that steps can be taken to minimize the risk associated with the hazard, such as minimizing the exposure or adding system safeguards to protect the system against such states.

The type of analysis required to guarantee consistency between the software requirements specification and the safety constraints depends upon the type of constraints involved. The presence of constraints can potentially affect most of the criteria that have been described in this appendix. Some types of constraints can be ensured by the criteria already described; others require additional analysis. For example, basic reachability analysis can verify that only safe states are reachable. Basic reachability analysis may need to be extended, however, to consider additional constraints on the *sequence* of events or states.

More generally, the specification may be checked for a general *safety policy* that is defined for the particular system. This process is very similar to checking that a specification satisfies a particular security policy. The following is an example of a general safety policy for which the specification could be checked:

1. There must be no paths to unplanned hazardous states.

   The computer never initiates a control action (output) that will move the process from a safe to an unplanned hazardous state.

2. Every hazardous state must have a path to a safe state. All paths from the hazardous state must lead to safe states. Time in the hazardous state must be minimized, and contingency action may be necessary to reduce risk while in the hazardous state.

   If the system gets into a hazardous state (by an unplanned transition that is not initiated by the computer such as component failures, human error, or environmental stress), then the computer controller will transform the hazardous state into a safe state (every path from a hazardous state leads to a safe state). The time in the hazardous state will be minimized to reduce the likelihood of an accident.

There may be several possible safe states, depending on the type of hazard or on conditions in the environment. For example, the action to be taken if there is a failure in a flight-control system may depend on whether the aircraft is in level flight or is landing.

3. If a state state cannot be reached from a hazardous state, all paths from that state must lead to a minimum risk state. At least one such path must exist.

If a system gets into a hazardous state and there is no possible path to a safe state, then the computer will transform the state into one with the minimum risk possible given the hazard and the environmental conditions, and it will do so such that the system is in a hazardous state for the minimum amount of time possible.

It may not be possible to build a completely safe system—that is, to avoid all hazardous states or to get from every hazardous state to a safe state. In that event, the system must be redesigned or abandoned, or some risk must be accepted. This risk can be reduced by providing procedures to minimize the probability of the hazardous state leading to an accident (such as minimizing its exposure time) or to minimize the effects of an accident.

E The SpecTRM-RL Models of CTAS at the DFW TRACON

Because of the size of the models, we have included only the graphical parts and not the transition tables.

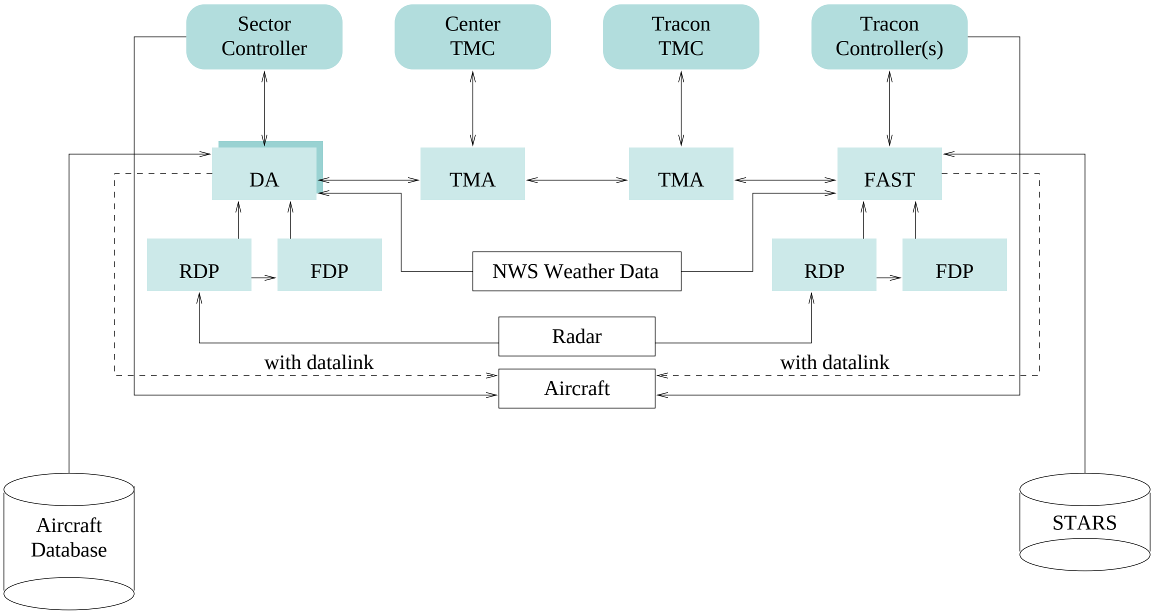


Figure 44: High Level commmunication diagram of CTAS components

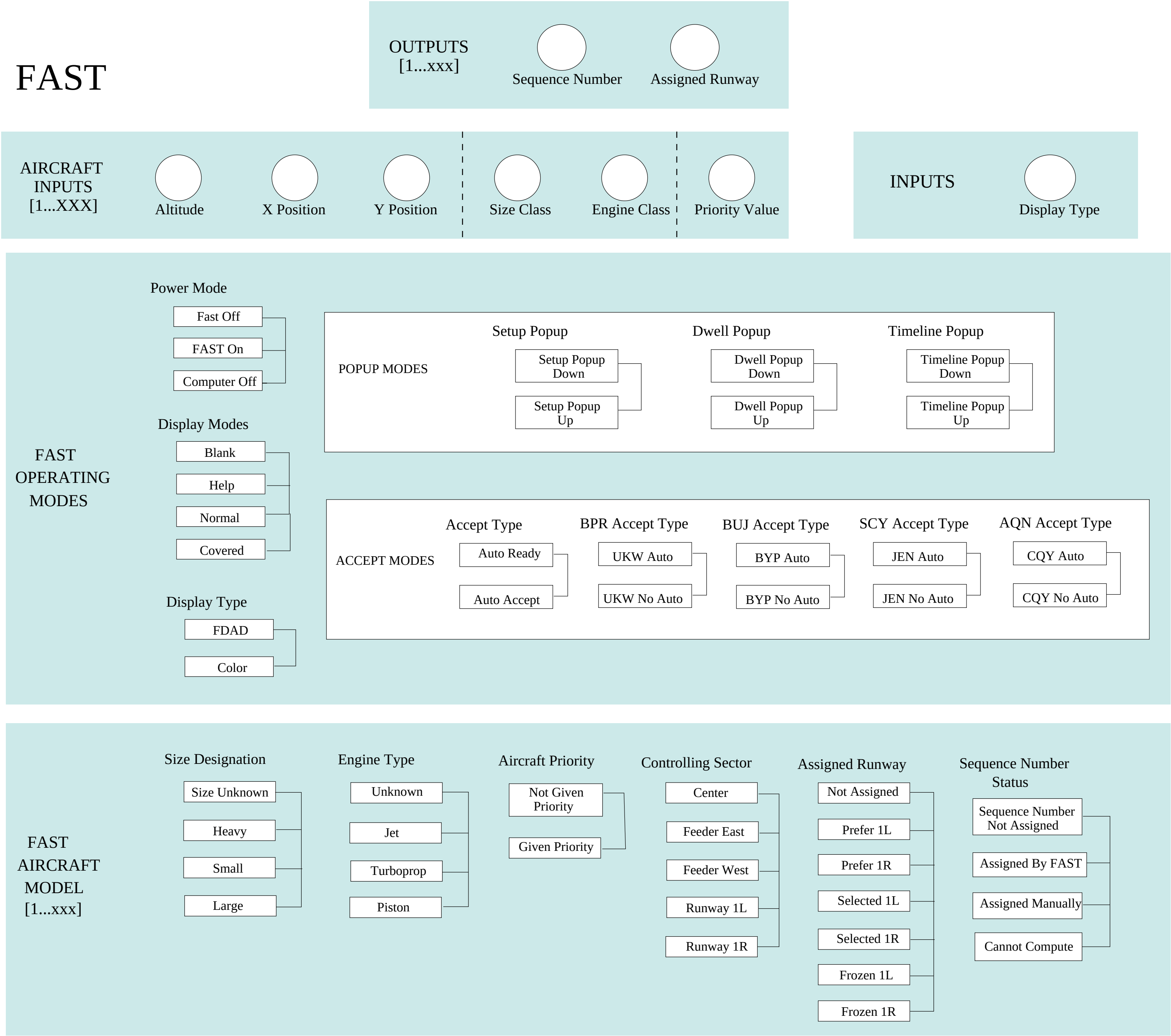


Figure 45: FAST Model

Descent Advisor

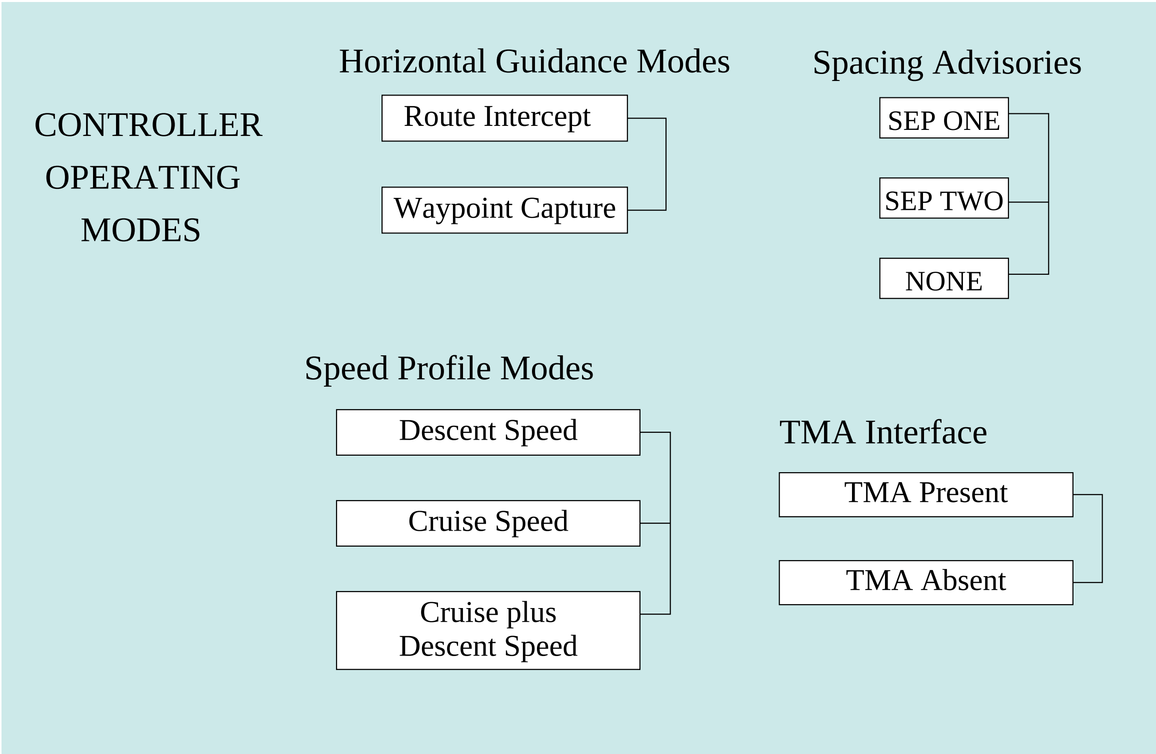


Figure 46: Descent Advisor Model

Descent Advisor

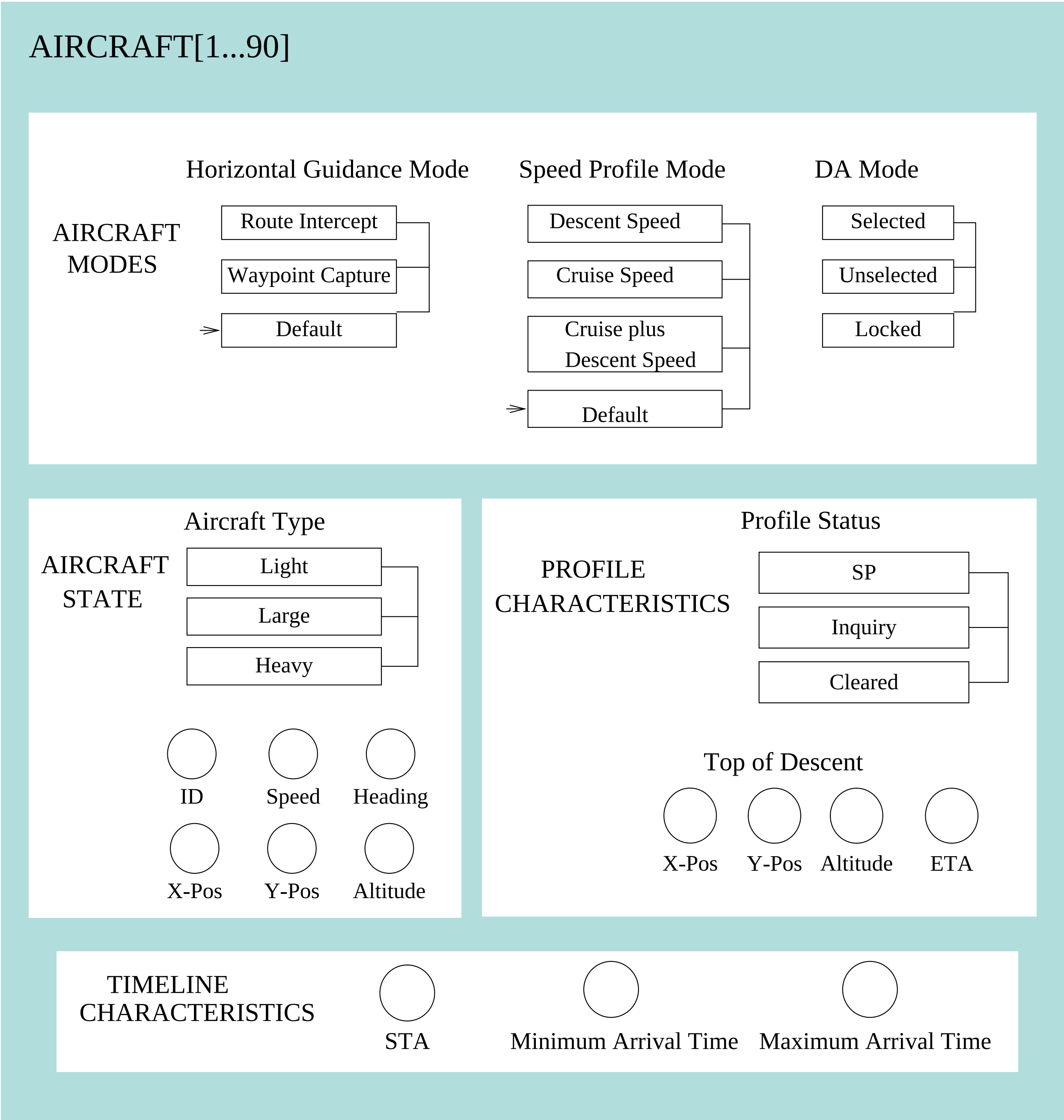


Figure 47: Aircraft [1 . . . 90] Model

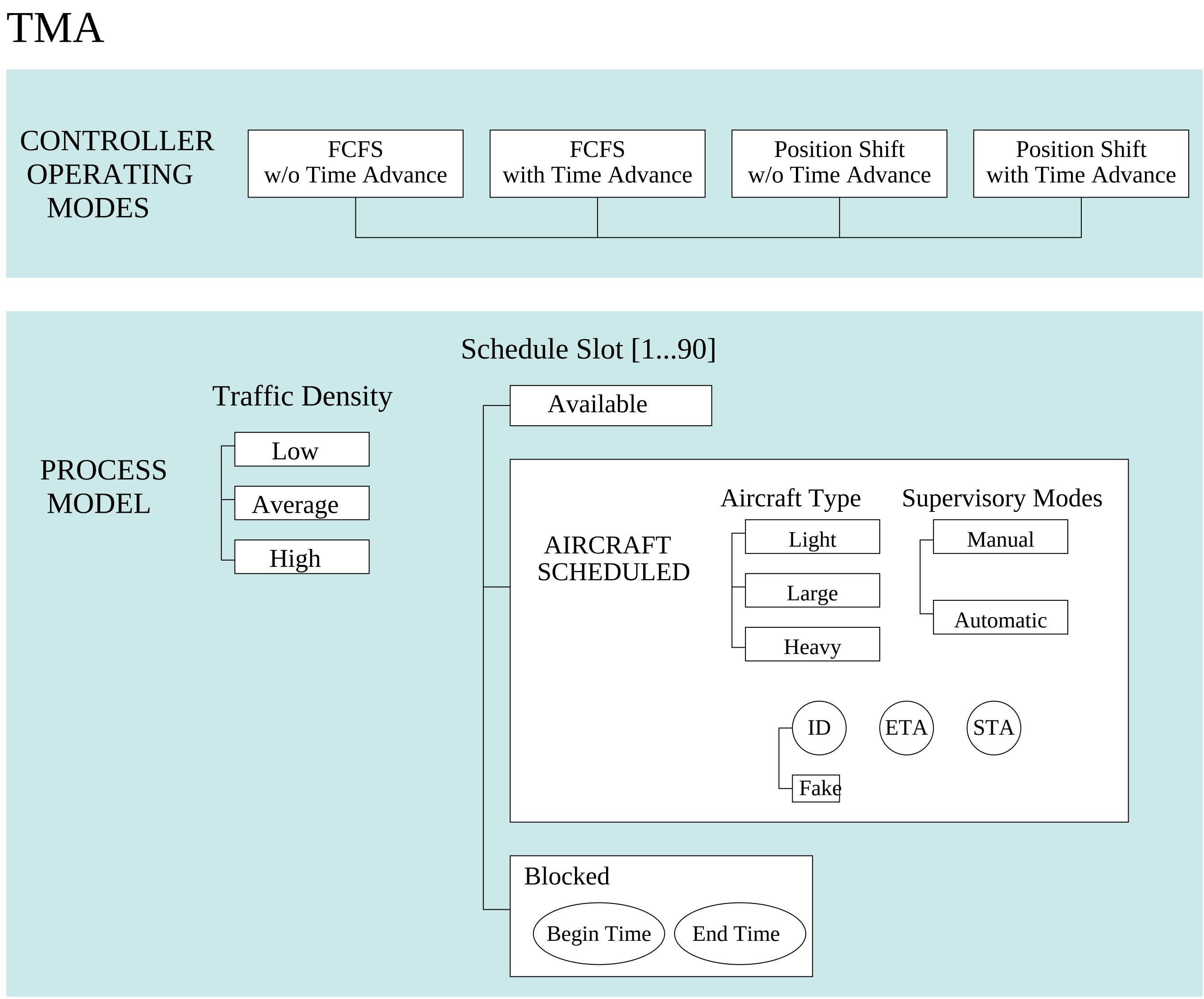


Figure 48: Traffic Management Advisor Model

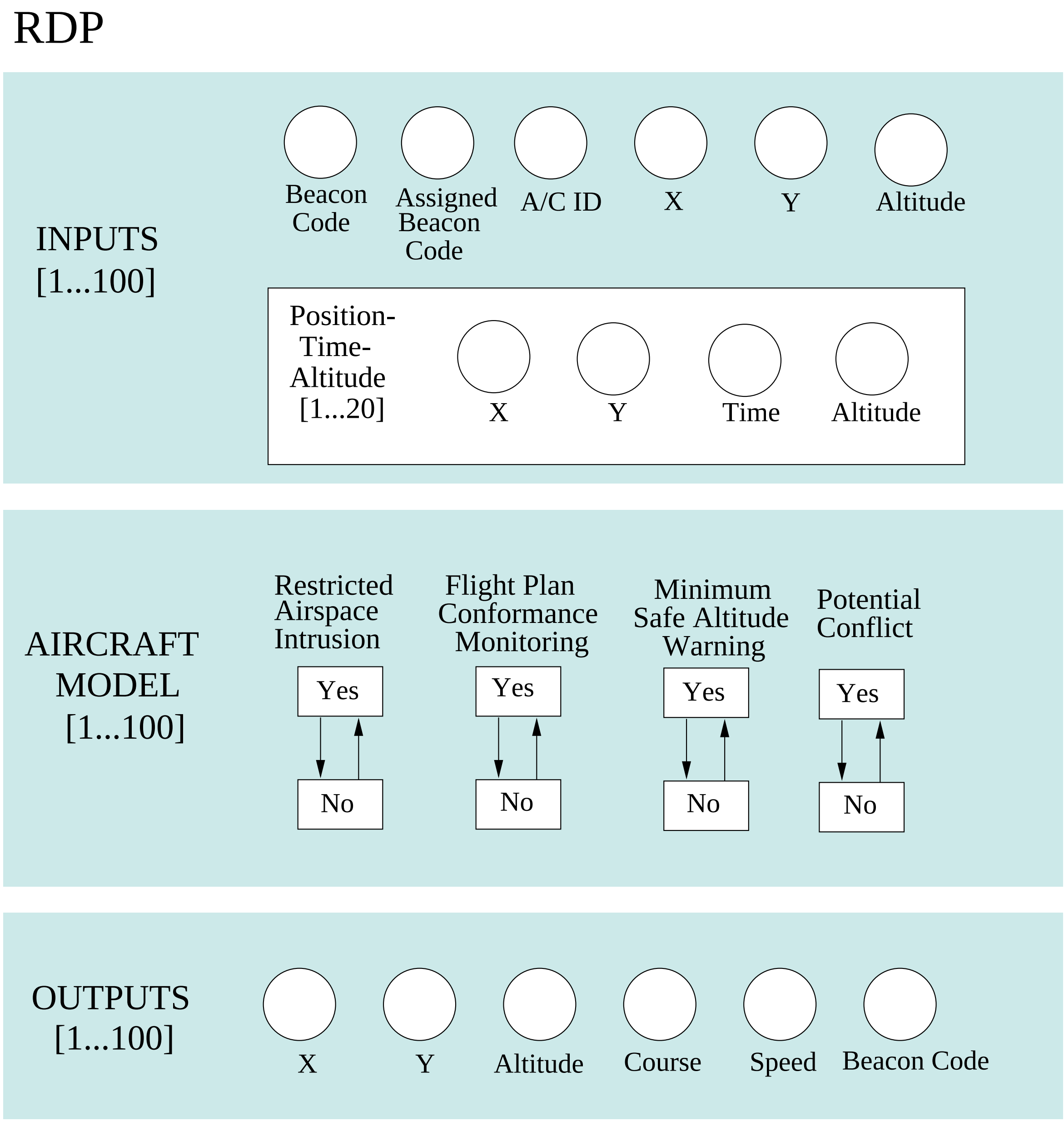


Figure 49: Radar Data Processor Model

F Partial Intent Specification of TCAS

Because the intent specification of TCAS is so large (750 pages, which includes the software design specification and much of the code), we have included it only in the printed final report for NASA. A copy of our example TCAS intent specification can be found online at: http://sunnyday.mit.edu/papers/intent.pdf