Integrated Safety Analysis of Requirements Specifications*

Francesmary Modugno, Nancy G. Leveson, Jon D. Reese
Kurt Partridge, and Sean D. Sandys

Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350

Abstract

This paper describes an integrated approach to safety analysis of software requirements and demonstrates the feasibility and utility of applying the individual techniques and the integrated approach on the requirements specification of a guidance system for a high-speed civil transport being developed at NASA Ames. Each analysis found different types of errors in the specification; thus together the techniques provided a more comprehensive safety analysis than any individual technique. We also discovered that the more the analyst knew about the application and the model, the more successful they were in finding errors. Our findings imply that the most effective safety-analysis tools will assist rather than replace the analyst.

Keywords: software safety, software safety analysis, software requirements specification.

Introduction

Although there are well-established techniques and procedures for analyzing electro-mechanical systems for safety, only relatively recently have researchers begun to develop similar techniques for software or for systems that contain software. The long-term goal of our research is to provide system developers with comprehensive support for software-safety analysis. To accomplish this, we must do more than simply propose techniques and demonstrate them on small examples, we also need to demonstrate the feasibility of applying them to realistic systems and to demonstrate their effectiveness. We can then apply what we have learned to improve the methods and tools, determine whether they can be effectively applied by others, and learn how to make them more usable by system developers.

Leveson and colleagues have been developing a series of techniques and tools to analyze safety-critical software [2, 8, 9, 10, 12]. We have demonstrated the techniques and shown that each can be effective individually for the analysis of safety-critical systems. For example, Heimdahl [2] found several sources of dangerous incompleteness and nondeterminism in the specification of TCAS II (Traffic Alert and Collision Avoidance System), an airborne collision avoidance system required on all U.W. aircraft carrying more than 30 passengers. To explore the feasibility of automated analysis, we also applied some prototype tools to an automated highway system model [14]. We did not, however, study the interaction between the analysis results nor did we study the analysis process itself.

To our knowledge, there has been no effort to apply all these techniques in an integrated safety analysis of a system or to compare the results of using different hazard analysis techniques on the same requirements specification. Moreover, there is little information on the analysis process itself. In this paper, we explore the feasibility of such a comprehensive safety analysis on a specification of a guidance system for a high-speed civil transport being developed at NASA Ames [5]. The goal of this case study was to determine the feasibility of performing such analyses and to evaluate the techniques and their contribution to the safety-analysis process. For example, we were interested in seeing how the results of the individual analyses interacted—what type of information each analysis provided and how that information complemented, was different from, or was supported by the results of the other analyses. We were also interested in learning about the analysis process: how difficult the techniques were to learn and apply, what problems we would encounter when learning and applying the techniques as a group, how difficult it would be to

*This paper will appear in the proceedings of the 3rd Int. Symposium on Requirements Engineering, Annapolis, Maryland, January 1997. The research described has been partly funded by NASA/Langley Grant NAG-1-1495, NSF Grant CCR-9396181, and the California PATH Program of the University of California.

interpret the results and integrate them, etc.

This paper reports the results of this empirical evaluation. First we provide a brief description of the guidance system. The model is much too large to include more than just a brief overview in this paper (the entire model is over a hundred pages long). More information about the specification can be found in the appendix. The main body of the paper presents an overview of software-safety analysis and the results of the analyses. We conclude with a discussion of our results and experiences along with planned future work.

The Guidance System

Figure 1 shows an overview of the Vehicle Management System (VMS) for a high-speed civil transport being developed at NASA Ames [5]. The VMS assists the pilot with tasks such as on-board flight planning, navigation, guidance and flight control. From the flight-path information received from the Air Traffic Controller, the Translator computes the Reference Flight Path (RFP), which completely specifies the aircraft's planned trajectory in space (i.e., its desired altitude, velocity, and flightpath angle, which is the number of degrees from the horizon in an earth reference frame) along with information such as the flap and gear extension control points, top of descent point and distance to next way point.

During automatic flight, the guidance system compares the aircraft's actual trajectory (as determined by sensor inputs) with the planned RFP trajectory and generates both lateral and vertical motion flight-control commands to null trajectory errors. These commands are sent to the Flight Control, which directly executes them, producing a change in the aircraft's velocity, altitude, or flightpath angle. During manual flight, the pilot can directly issue commands to the Flight Control in order to produce the desired effect.

The guidance system can also be operated jointly by the underlying computer and the pilot. In this mode, the pilot can define the reference flight path by entering the desired altitude, velocity and flightpath angle into the Mode Control Panel (MCP) and can select the operational mode of the guidance system by depressing the appropriate mode buttons on the MCP.

During both automatic and joint flight modes, the guidance system selects its operational mode based on (1) the pilot's selections from the Mode Control Panel, (2) the aircraft's position and velocity relative to the desired trajectory, and (3) the previous operational mode. Using this information, it computes the desired flight control commands, which include the commanded flight path angle and the commanded thrust. The guidance system specification is described briefly in the appendix.

Types of Safety Analyses

A *safe system* is one that is free from accidents or unacceptable losses [9]. The heart of analyzing a system from a safety perspective is identifying and analyzing the system for *hazards*, which are states or conditions of the system that combined with some environmental conditions can lead to an accident or loss event [9]. Once the hazards are identified, steps can be taken to eliminate them, reduce the likelihood of their occurring, or mitigate their effects on the system. 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.

A hazard analysis requires some type of model of the system, which may be an informal model in the mind of the analyst, a written informal or formatted specification of the system, or a formal mathematical model. Different models allow for different types of analyses and for additional rigor and completeness in the analysis. For the system evaluated in this paper, we use a state-machine model of the system's required black-box behavior. We model the requirements instead of an actual software design because many accidents involving software systems can be traced to requirements flaws [13]. In addition, by modeling and analyzing the requirements, we can find problems early in the development cycle when they are more easily and effectively addressed.

A fundamental tenet of linear control theory is that every controller is or contains a model of the controlled process. In our specification language, this model describes how the controller will behave with respect to various state changes in the controlled process. It is also used to determine the current process state given the previous state and new sensor measurements (readings) of various process variables. Because a model is an abstraction, it is necessarily incomplete.

Hazards and accidents can result from mismatches between the software view of the process and the actual state of the process—that is, the model of the process used by the software gets out of synch with the real process. This mismatch may occur because the internal model is incorrect or incomplete with respect to important properties or the computer does not have accurate information about the process state.

Safety then depends on the completeness and ac-

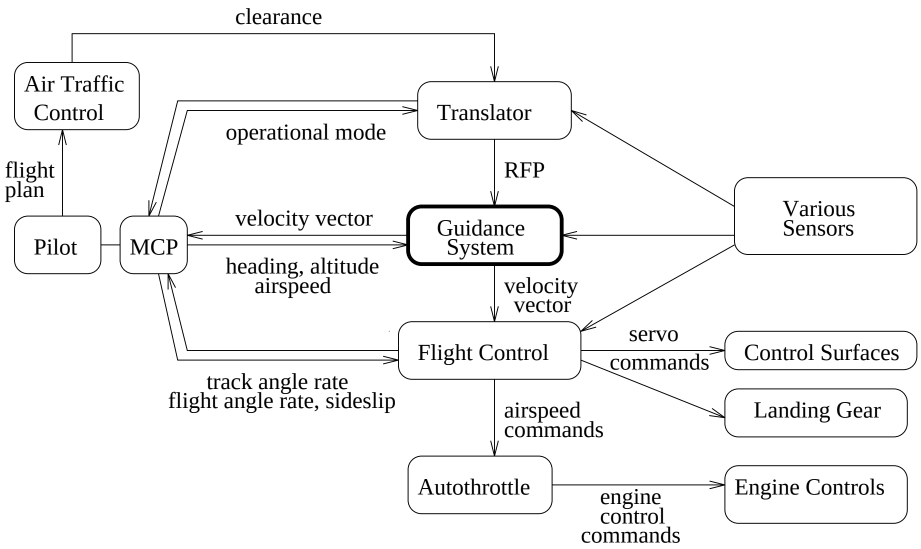

Figure 1: Overview of the Vehicle Management System for a high-speed civil transport being developed at NASA Ames. At the center is the guidance system, which automatically controls the aircraft's flight.

curacy of the software's model of the process. A state-machine specification of system or software requirements explicitly describes this model and the required behavior of the software. A goal of software-safety analysis is to ensure that the model of the controlled process (i.e., the requirements specification) is sufficiently complete that it specifies safe behavior in all circumstances in which the system will operate. Jaffe [7, 8, 9] has defined general criteria for determining whether such models satisfy this goal. The results of applying these criteria to our specification are described below.

The specification (model) can also be analyzed with respect to specific, known hazards. We accomplish this goal using what we call *State Machine Hazard Analysis* (SHMA) [9]. SMHA, like most safety analyses, involves some type of search. How that 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 [9].

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.

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 reasonable set 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. Tracing an initial event forward will most likely result in multiple 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 this type of search is that hazards that have not previously been identified can theoretically be found.

Tracing backward from a particular hazard or accident to its preceeding states or events may uncover multiple initiating or contributing events, but the hazards usally must be known. 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.

In this case study, we performed both forward analyses (simulation and deviation analysis) and a backward analysis (fault tree analysis), as described below.

Analyzing the Guidance System Model

Errors were found when constructing the model, checking the consistency and completeness criteria, and performing the forward and backward analyses. We note that the NASA guidance system specifications we used are in the research stage and still under development. The fact that we found errors or potential hazardous states reflects only on the preliminary nature of the specification.

Knowing the backgrounds of the modelers and analysts is helpful in interpreting the results. Modugno, who did the bulk of the model construction as well as overseeing the analyses and synthesizing the results, is a computer scientist with no physics training. Within computer science, her background is human–computer interaction and only recently has she begun to work in software safety. Reese is trained as a computer scientist and created one of the forward analysis methods [15]. Sandys and Partridge are Ph.D. students in computer science, studying software safety. In addition, Sandys holds a bachelor's degree in physics and helped construct the parts of the model that required physics knowledge. Finally, Leveson has been working in the area of safety for 15 years. She helped with the initial stages of model development although she did not help perform analyses in order to determine how well they could be done without her expertise.

The original guidance system specification was written by a NASA expert in aircraft control and guidance systems. He has a Ph.D. in physics, is a licensed pilot, and has been working in this area for over 30 years. We interacted with him while constructing the model, and he provided some support during the analyses, especially in determining appropriate environmental data to use in the formal simulation of the model. The next sections describe the errors found during model construction and analysis.

Errors Found While Constructing the Model

The specification was developed using three different documents provided by the NASA Ames researchers: 1) a description of the overall goals of the guidance system design along with a description of the vertical operation modes using a combination of English, Laplace diagrams and state machines; 2) a Jackson Charts [6] specification of the vertical operation modes; and 3) pseudo-code used to create a program to control a vertical motion simulator for pilot testing of the guidance system.

In the process of constructing the model, we found errors in the original guidance system specification. Discovering these errors points to the utility of going through the process of developing a model of specifications as well as to the utility of the model itself. We note again, however, that these documents are still in development so many of the errors we found might have been found in a later stage by the developers.

For example, we found a line in the pseudo-code in which the system transitioned to one state but illuminated a different button on the display. This error was probably due to quick editing on a word processor when writing the pseudo-code (i.e., copying a portion of the code and editing it incorrectly). Nonetheless, a programmer might not notice the error. Indeed, we might not have noticed it either had our modeling language not forced us to think about the relationship between the computer logic and the display information; that is, the form of our model caused us to think about the requirements specification in a particular way.

As a more serious example, we found places in the pseudo-code in which the system prompted the pilot for a particular data value when in fact another value was required by the guidance system logic. Again, we discovered this error because, when constructing the model, we had to trace the flow of the pilot's inputs through to the computer logic.

We also found several omissions in the NASA specification when constructing the model. For example, nowhere in the documents did it say how often the inputs would arrive. We obtained this information from conversations with the designer as we were constructing the model and attempting to understand the logic of the guidance system. The notations used for the original NASA specifications did not require this information to be included.

Note that these errors were not necessarily found because of any mathematical notation in our language—in fact, many in the formal methods community would not consider our language to be "formal." We believe that we found these errors in translating from the Jackson Charts and pseudo-code into our modeling language (called Requirements State Machine Language or RSML) mainly because the process required that we consider how to represent each Jackson Chart action or line of pseudo-code in RSML and then determine where it fit into our model. The type of thinking and analysis required to construct a state-machine model provides a basis for uncovering some types of requirements errors. The same type of benefit may be found using other modeling techniques, but will depend upon the modeling methodology and language and is not automatically a property of all models.

In summary, the form of our RSML model (i.e., the state decomposition we decided on), the process of constructing the model from the various informal specifications provided by the NASA Ames researchers, and the resulting structured model helped us to discover errors in the specification.

Completeness and Consistency Checks

Jaffe and colleagues [8, 9] have defined a set of formal criteria to identify missing, incorrect, and ambiguous requirements for process-control systems. Briefly and informally, the criteria ensure (1) completeness of transitions and default values during normal and non-normal operation, including startup and shutdown; (2) complete specification of all inputs and outputs; (3) complete specification of the interaction between the computer and the operator; (4) complete description and handling of all inputs including essential value and timing assumptions about these inputs; (5) complete specification of the output conditions with respect to timing and value, including environmental capacity, data age, and latency requirements; (6) complete specification of the relationship between inputs and outputs, including feedback loops and graceful degradation; (7) and complete specification of the paths between states with respect to desirable properties such as basic reachability, recurrent or cyclic behavior, reversible behavior, reachability of safe states, preemption of transactions, path robustness, and consistency with required system-level constraints.

Analyzing a specification in terms of these criteria depends on the form of the specification (its size, language, etc.). In some cases the criteria can be enforced by the syntax of the specification language, while in other cases, the criteria can be checked by manual inspection or with the assistance of automated tools. For example, Heimdahl [2] has automated the checking of RSML specifications for two of the 47 criteria, i.e., those to ensure robustness and nondeterminism. Heitmeyer and colleagues [3] provide tools similar to Heimdahl's to check for the internal consistency of specifications in SCR [4], a state-based specification language that uses an assortment of tabular notations to define state transitions.

Without realizing it, Modugno had begun to do some of these checks informally when constructing the model and later when examining the completed model. The syntax of the modeling language is designed to make some omissions obvious. She later realized that her efforts in examining the model in this way were really a part of the planned completeness criteria checking. We also found errors relating to completeness by performing forward simulation and the other hazard analyses.

Some of the errors that were apparent immediately from examining the model involved missing transitions. For example, by reviewing the RSML graphical representation of the guidance system, we noted that several components had states with no transitions between them. Upon further examination, we discovered there was no detailed specification for these transitions in the documents. As a specific example, once both the Glideslope Lock and Altitude Lock were set, they were never unset, i.e., in examining the model we noticed that there was a transition from the state Glideslope-Lock-Off to Glideslope-Lock-On and from Altitude-Lock-Off to Altitude-Lock-On, but there were no transitions in the other direction. This type of omission is defined by one of our path completeness criteria, i.e., *reversibility*.

In addition to missing transitions, examining the model revealed input not used and missing output. When constructing the model, we had created an input (output) interface for each input (output) in the pseudo code. A simple search on each variable name helped us determine if and how each input was used or output was produced. Several inputs were never used, and two output values were never produced.

After completing the model, Modugno manually examined it for violations of each of the criteria. She believes that her familiarity with the model (from having constructed it) greatly facilitated the analysis. We have not yet had an avionics expert try to apply the criteria, but plan to do so soon.

Several omissions were detected during the manual check of the completeness criteria. For example, the specification for the Primary Flight Display did not detail how the displayed geometric figures mapped to actual data values, how long they were to be displayed and what triggered them to appear or disappear. Similarly, as we noted above, some of the mode annunciators were missing trigger events. Both these omissions were found by analyzing the specification using the human–computer interface criteria.

Several incompletenesses in the state specification were found, among them uninitialized input variables, such as measured speed and measured altitude (which are necessary to model the system during take off), and unspecified timing intervals between inputs. The NASA documents did not specify how often the sensors would report their data values nor was there any specification on how long the system should wait for the pilot's input once the prompts appeared, although a timeout was implied in the Jackson Chart and pseudo-code specifications.

Additionally, we found several states that did not specify a response to certain inputs while in that state. For example, if the pilot selects a particular vertical guidance mode, the system will prompt him or her to enter the reference speed and altitude. However, there was no specification of the system behavior if the pilot then selected another vertical guidance mode without first entering the requested data.

With respect to robustness, the current NASA specification for the guidance system contains no checks for legal ranges or timing values and thus no behavior is specified for out-of-range input values or late, early, or missing inputs. Similarly, range and timing values were missing for flight control command outputs along with data age limits (the length of time before input data or output commands are considered obsolete).

Finally, by examining the relationship between the outputs and inputs, we discovered that there were no feedback loops to check on the response of the aircraft to the generated flight control commands, and thus no behavior was specified to respond to either expected input values or unexpected, early, late or missing input values. These omissions could cause a failure in the system to go unnoticed.

The NASA guidance system is still under development and many of these omissions probably would have been caught and corrected without our analysis. However, these types of requirements specification flaws are common and often persist until late in development or actual use of the software. For example, Lutz found that the Jaffe completeness criteria covered most of the 192 requirements errors identified as safety-critical that were not detected until system integration testing of the Galileo and Voyager spacecraft [13]. We found manual checking of the criteria to be helpful in finding important specification omissions and believe that automated tools to assist the analyst in checking the criteria might detect errors not found by our manual process.

Forward Simulation

Forward simulation was performed by Modugno and Sandys. The RSML simulator was used to execute several scenarios to observe the states of the controller given specific inputs. The extensive physics background of Sandys along with help from a NASA expert were required to generate reasonable scenarios. The quality of the results of the simulation will, in general, be limited by the quality of the test scenarios.

The depth-first search property of forward simulation makes using simulation alone infeasible to detect reachable hazardous states. If, as is reasonable, hazards are assumed to be infrequent, then the probability of blindly finding a hazard using this type of search is low. However, forward simulation can be useful when used in conjunction with other analysis methods. For example, we found we could use other analysis techniques to narrow the search space and then use forward simulation to investigate the smaller region.

Despite these limitations, by simply tracing the transitions forward for individual components, we did find a hazardous state within the normal operation of the requirements specification. The state was one in which the flaps were not DOWN but the plane was in a landing approach. During landing, the flaps help slow the plane. In certain instances, the plane could get "stuck" in this hazardous state, potentially leading to an accident. The problem arises from an assumption in the specification that inputs triggering the flap motion will arrive in a particular order. In particular, the specification assumes that the information about the point in space where the pilot is supposed to initiate the flap extension (FLAP-INIT-POINTS) will arrive before the boolean input indicating that flap extension can begin. Yet there is no indication of this assumption anywhere in the specification. Specifications should either clearly state such assumptions, or they should be written to handle their violation.

Uncovering this assumption revealed another assumption within the FLAPS component—namely that the FLAP-INIT-POINT input will arrive before the plane reaches that point in space. Nothing in the specification indicates what should happen if the FLAP-INIT-POINT information arrives after passing it. In fact, a similar situation may have contributed to the recent crash of a Boeing 757 in Cali, Columbia. In that accident, the pilots entered a way point into the guidance system that the aircraft had already passed. In an attempt to reach that way point, the guidance system commanded the plane to turn around, causing the aircraft to crash into a mountain.

These two assumptions are related to some of the errors revealed by the completeness and consistency checks. Theoretically, they are covered by the completeness criteria, but we did not find them during our manual inspection of the model using the criteria as a checklist, pointing to the need for better manual inspection methods or automated assistance to the analyst.

Deviation Analysis

Deviation Analysis is a new type of forward search technique that takes its inspiration from HAZOP (HAZards and OPerability analysis), a very successful analysis procedure used in the chemical process industry. Both techniques are based on the underlying system theory concept that accidents are the result of deviations in system variables.

Deviation analysis takes a formal model of the system along with deviations in the system inputs from their normal or expected values and examines the effects of these deviations on the system's behavior. It can also help identify potential system hazards resulting from deviations in system inputs.

To assist the analyst, Reese [15] has developed a tool for performing a deviation analysis on an RSML specification. The analyst first selects one or more inputs to deviate and describes how to deviate them (high, low, very high, very low, and so on). The analyst then marks states, functions or outputs for which a deviation is considered hazardous. The tool produces a set of scenarios in which the deviated inputs would produce a deviation in a marked state, function or output. Each scenario lists assumptions on other input values that lead to the deviation and what effect the initial deviations and additional assumptions have on the affected state, function or output. For example, an input that is deviated high could cause the system to enter a hazardous state by, for example, a function computing a value that is too high or the software producing an incorrect output.

The deviation analysis was performed by Reese. By deviating different inputs, he found several potential hazards in the model. For example, he wanted to determine what effect a too-high measured speed would have on the commanded flight path angle (recall that the measured speed is the speed of the aircraft as determined by the sensors, and the commanded flight path angle is one of the guidance system outputs—see Figure 1). Using the deviation analysis tool, he deviated the measured speed "high" and marked the function that computes the commanded flight path

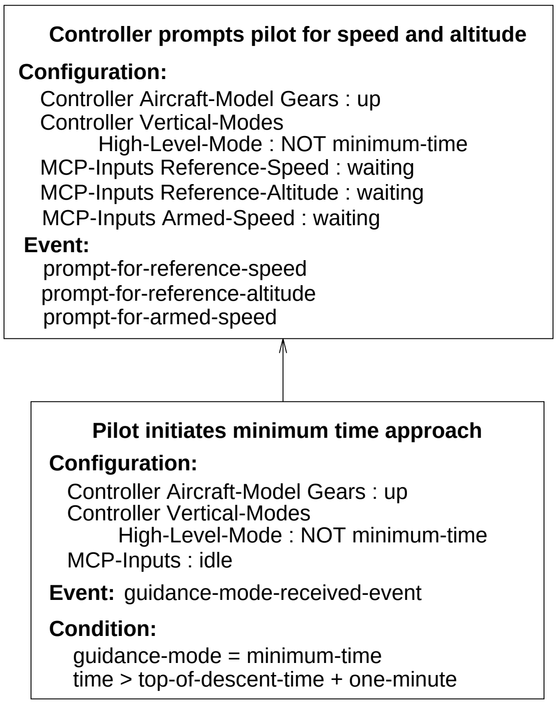

Figure 2: The sequence of events leading to the hazardous state in which the plane is in minimum-time mode and the landing gear are not deployed is initiated by the pilot selecting the minimum-time approach button when the plane is more than one minute away from the top-of-descent time.

angle. The deviation analysis tool initially produced five scenarios. Each scenario showed that deviating the measured speed alone would not cause a deviation in the commanded flight path angle. However, they also indicated that the initial deviation, coupled with one or more other deviations in the input, would cause the commanded flight path angle to be incorrect.

In performing the deviation analysis, Reese was limited by his unfamiliarity with the guidance system model. He was initially unable to determine which states or functions to mark as important items to analyze when deviating a particular input. Only after understanding the model in detail was he able to interpret the scenarios output by his tool and use them to investigate other deviations. His experience supported our observations thus far of the importance of application knowledge in performing the analysis.

Backward Search

In backward search, the analysis starts with a hazardous state and builds trees showing the events that could lead to this state. In order to perform this analysis, we first had to come up with a list of potentially hazardous states. Using these configurations, Modugno then did a backward search, examining the results for potential hazards under expected and failure behaviors. She found the process to be labor intensive and required in-depth understanding of the model. In addition, the trees grew quickly and the presentation quickly became unwieldy. This same problem has been noted with other types of automated analysis applied to hardware models [1].

As an example, consider the hazard of the landing gear not deploying during a minimum-time approach landing. The top node of the generated tree thus represents the case in which the system is in the minimum-time landing mode and the gears are up.

There are two changes that can lead to this configuration—a change in the gear position from DOWN to UP or a change in the vertical mode from any other mode to minimum time. By analyzing the state transitions, we determined that the only way the gears can go from DOWN to UP is if the mode changes from minimum time to take-off/go-around. Therefore, the only way the guidance system can get into the hazardous state is when the gears are already up and the plane enters minimum-time mode. Because there were a large number of potential paths to the hazardous state, we used forward simulation to help prune the number of paths that needed to be considered.

For the guidance system to transition to minimum-time mode, the pilot must initiate a minimum-time approach by selecting the minimum-time mode button on the MCP and then entering the reference speed, reference altitude, and armed speed into the MCP within a certain time period. Figure 2 is a portion of the tree showing the partial configuration of the guidance system that gives rise to this series of events: the gears are up; the Controller's high-level vertical mode is not minimum time; and the MCP Inputs from the pilot are idle (i.e., no input is expected). The figure also shows the event and conditions under which the guidance system can transition out of this configuration. The event that triggers the transition is a `guidance-mode-received-event`, which indicates that the pilot has selected a new vertical guidance mode. The conditions that must be satisfied for the transitions to fire are that 1) the selected guidance mode must be minimum time, and 2) the selection must be made more than one minute before the top-of-descent time. If these two conditions are met, then the guidance system will prompt the pilot to enter the reference speed, reference altitude and armed speed.

Hence, there is a normal sequence of events that can lead to the hazardous state. By analyzing the specification, we found three conditions under which the system will remain in this hazardous state under normal operation (which means that the plane will land without the gears extended). For example, for the gears to transition from UP to DOWN, the measured speed must be 250 knots. If that exact speed is never reported by the sensors, the transition will never occur. Using this analysis, we were able to change the conditions under which the gears UP to DOWN transition occurs to avoid remaining in the hazardous state. Similar analyses and changes addressed the other hazardous conditions.

Discussion and Conclusions

We have described the results of an empirical evaluation of a set of hazard analysis techniques. Several conclusions follow from our results.

First, these techniques can be applied to real systems. We were able to build a model of a complex system and to apply completeness criteria and forward and backward hazard analysis techniques to that model. We have shown that these techniques work on more than very small, research-paper examples. In the process of demonstrating this, we obtained additional insight into the analysis process, which we will use to improve the techniques in the future.

Second, the techniques were able to find real errors in the specification. Again, we note this does not imply anything about the quality of the NASA specifications. The project developing the guidance system is a research project, and we were modeling early designs for the system. The emphasis in this NASA project has not been on getting complete specifications, but rather in examining ways to reduce modes and hence mode confusion in flight management computers. However, the errors or omissions we found are typical of those found in real system specifications and designs, including errors that have been found to have contributed to accidents in the past. Our goal was to determine whether our techniques could find important errors in real systems, and they did.

We also do not want to overstate the results. The process of carefully reviewing any specification or attempting to understand a system specification well enough to model it is likely to uncover errors, no matter what techniques are used. We cannot determine how effective our techniques are, especially with respect to alternatives, without more careful comparisons and perhaps controlled experiments.

Third, we found that the various hazard analysis techniques provided us with different and complementary types of information. While checking the completeness criteria manually, we found some potentially important omissions in the system specification. In the automated forward analyses, we detected some initial states and events that could lead to hazardous system states. In the backward analyses, we started with hazardous states and conditions and identified several possible paths leading to those hazards. All these types of analysis were useful. The backward analyses forced us to think about potential hazards of the system and examine their causes. The forward analyses forced us to think about potential failures and examine their effects. Together the analyses provided us with a more complete safety analysis than any of them alone provided.

Some of the analysis techniques support other ones. For example, in the fault tree analysis, taking a step back from an initial configuration provides a set of possible previous states. By performing a forward simulation from each of these states, we were able to prune away those states that cannot really precede the initial configuration. Similarly, both forward and backward simulation assisted in the checking of some of the completeness criteria.

Fourth, the design of the modeling language can support or hinder the analysis process. We discovered a number of important errors in the NASA documents simply by constructing the state-machine specification. We credit this result to the type of thinking and analysis required when constructing this type of model.

In addition, we believe this particular representation facilitated the analysis process; the specification provided a structure that supported the different techniques. For example, the organization of the model provided an orderly way to examine the specification for completeness and consistency errors. Also, the state abstraction and organization facilitated the fault tree analysis: Having transitions between states and the conditions under which those transitions are taken located together makes tracing the events backwards through the specification simple. In both the Jackson Chart specification and the pseudo-code, this information is dispersed throughout, making the backward tracing of events and conditions difficult. For example, often the conditions for a particular state change in the pseudo-code are buried within the conditions of a group of nested `if-then-else` statements and function calls. Tracing these back could be difficult (and we note, was difficult and error prone; back tracing through function calls and nested `if-then-else` statements is how we constructed the model initially).

In addition, the fact that a subtle typographical error in one specification language translates to an obvious omission in another language suggests that analysts can be provided with specification languages that focus their attention on the places where we have found that errors are likely to be made. The same can probably be said of analysis—the form of the analysis results will most likely affect the ability of analysts to interpret them with respect to their own application expertise and knowledge.

A conclusion that might be drawn here is that the design of a modeling or specification language should reflect the type of analyses to be performed on it, i.e., an understanding of the type of analyses desired should precede language design. In addition, many if not most errors will be found during expert review rather than by automated tools, so readability and organization of the specification to enhance the ability of the expert to find errors are also important considerations in specification language design. Our modeling language (RSML) is changing and evolving as we learn more about the modeling and analysis process.

Finally, we found that using the hazard analysis techniques requires application expertise, and hence they are probably best applied by application experts. When we began this project, none of us had any experience with guidance systems. In addition, the main analyst (Modugno) was new to the area of software safety and had no prior experience with the techniques. Although we found that the techniques were easy to learn, applying them successfully depended on an understanding of the application domain. Even building the model required a detailed understanding of the physics of the plane's motion. Moreover, translating from the designer's notation into RSML required an understanding of engineering notation not commonly used in computer science.

Whether application experts can use our modeling and analysis techniques and will find them useful, however, is yet to be determined. We have ascertained the feasibility of performing an integrated safety analysis and the utility of such an analysis. Our next step is to determine whether others can use the techniques to perform an analysis. Then we can begin to determine what tools we need to develop to support analysts.

Our findings so far suggest that the most effective safety-analysis tools will assist rather than replace the analyst. We should therefore focus on building tools that augment human abilities rather than attempting to do the analysis completely with tools alone. For example, because both the backward and forward analysis techniques require searching through a large space, tools to help the analyst prune that space and navigate through it could be helpful. Conversely, expert knowledge of the application allows the analyst to select the most plausible search paths and to eliminate immediately scenarios that are physically or logically impossible in the system being analyzed. We hope to have a NASA guidance system expert apply our techniques and tools to compare the results obtained in terms of number and type of problems found. A partnership between application, safety, and tool experts will most likely turn out to be the most effective way to produce useful results.

We plan to study further the whole issue of tool design. For example, the type of support that automated tools and techniques should provide will depend on an understanding of the cognitive demands of the particular analysis techniques and an understanding of how automation can be used to lessen those demands. Studying the process of how experts learn and use the analysis techniques should help us gain some of this understanding, as well as using results of research in other fields such as cognitive psychology and cognitive engineering.

We also plan to extend the modeling and analysis techniques to include ways to model the human operator and to provide techniques for human-error analyses. As computers have become more sophisticated and their role in process control has changed from simply an interface to the process to an autonomous controller, a large number of accidents in safety-critical systems have been blamed on "human error." We need to understand why these errors occur and how we can prevent them. Modeling the operator and analyzing the entire system—automated controller *and* operator—for potential hazards can help us reach this goal.

Acknowledgments

We would like to thank Charlie Hynes at NASA Ames for all his help and for providing the NASA guidance system specifications.

References

[1] P.K. Andow, F.P. Lees, and C.P. Murphy. The Propagation of Faults in Process Plants: A State of the Art Review. *7th International Symposium on Chemical Process Hazards*, University of Manchester, 1980.

[2] M.P.E. Heimdahl and N.G. Leveson. Completeness and Consistency Checking of Software Requirements. In *IEEE Transactions on Software Engineering*, vol. 22, no. 6, June 1996.

[3] C.L. Heitmeyer, B.L. Labaw, and K. Kiskis. Consistency checking of SCR-style requirements specifications. In *Proceedings of the International Symposium on Requirements Engineering*, 1995.

[4] K.L. Heninger. Specifying software for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, 6(1):2–13, January 1980.

[5] C. Hynes. An example guidance mode specification. Technical report, NASA Ames, 1995.

[6] M.A. Jackson. *Principles of Program Design*. Academic Press, 1975.

[7] M.S. Jaffe. *Completeness, Robustness, and Safety of Real-Time Requirements Specification*. Ph.D. Dissertation, UCI, June 1988.

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

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

[10] N. Leveson, S. Cha, and T. Shimeall. Safety verification of ada programs using software fault trees. *IEEE Software*, 8(7):48–59, 1991.

[11] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. Requirements Specification for Process-Control Systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, 1994.

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

[13] R. Lutz. Targeting safety-related errors during software requirements analysis. In *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1993.

[14] V. Ratan, K. Partridge, J.D. Reese, and N.G. Leveson. Safety analysis tools for requirements specifications. *Compass 96*, Gaithersburg, Maryland, June 1996.

[15] J.D. Reese. *Software Deviation Analysis*. Ph.D. Dissertation, UCI, 1996.

Appendix: The Guidance System Specification

We specified the control behavior of the guidance system using a parallel state-machine model. The modeling language is essentially RSML [11], which we previously developed to specify an aircraft collision-avoidance system, although the language is evolving as we gain more experience in modeling real systems and determine what is required and desirable to build understandable and analyzable models.

In our modeling language, components are modeled by parallel state machines. Each state machine is composed of states connected by transitions. Transitions are triggered by input or by internal events and are only taken when their guarding condition is true. A triggered transition produces an output or an internal event, which can in turn trigger transitions in other state machines.

In addition to the graphical specification of each state machine, the RSML model defines the guarding conditions on a transition using a logic table (AND/OR table). Figure 3 shows an example. The far-left column of the AND/OR table lists the logical phrases. Each of the other columns is a conjunction of those phrases and contains the logical values of the expressions. The table evaluates to true if one of its columns is true. A column evaluates to true if all of its elements are true. A dot denotes "don't care."

As shown in Figure 4, the guidance system can be modeled using three parallel state machines: the MODE CONTROL PANEL, which models the pilot's selections on the Mode Control Panel; the DISPLAYS, which models the state of the displays; and the CONTROLLER, which models the software logic.

Controller. The CONTROLLER issues commands to control the lateral and vertical motion of the plane. The particular command issued is dependent on the current CONTROL MODE of the guidance system, which indicates whether the guidance system is in fully automatic mode or jointly (pilot and computer) controlled mode, the current LATERAL MODE, the VERTICAL MODE, and the controller's model of the state of the aircraft.

For space reasons, we describe only the VERTICAL MODES state machine of the CONTROLLER, which is used to specify the conditions for issuing flight commands that control the vertical motion of the aircraft (Figure 5).
The guidance system has four High-Level Modes:

- `Minimum Time`: engaged to minimize time during a landing,
- `Minimum Fuel`: engaged to minimize fuel during a landing,
- `Take Off/Go Around`: engaged when a landing is missed, and
- `Standby` : engaged when no other vertical high-level mode is active.

These modes control the aircraft via the `Intermediate Modes`:

- `Altitude`: capture and hold a particular altitude, and
- `Glideslope`: track and capture the glideslope, which is a line through space relative to the ground.

The `Intermediate Modes` achieve their goals by interacting with the `Low-Level Modes`, which can be initiated only by the guidance system (that is, the pilot cannot directly instruct the guidance system to enter one of these modes). The three guidance-system-initiated modes are `climb`, `level flight`, and `descend`.

The appropriate vertical operational mode is determined using inputs from the environment (such as the Reference Flight Path Table and input from the sensors) as well as the aircraft model (see Figure 4). The Aircraft Model includes process state information, such as the position of the gears and flaps.

Mode Control Panel. The MODE CONTROL PANEL, which models pilot inputs to the guidance system, is composed of four parallel state machines that model the control, lateral and vertical mode buttons that the pilot selects on the MCP, and other input data. Again for space reasons, we describe only the vertical mode selections.

There are six buttons on the MCP that the pilot can select to control the vertical modes of the aircraft, as described above: minimum-time, minimum-fuel, take off/go around, standby, altitude, and glideslope .

The first four buttons represent the pilot selection of a high-level vertical mode, while the last two represent selection of an intermediate vertical mode.

Displays. The final component of the guidance system is the `Displays`. We model the information on two of the aircraft displays: the MCP and the Primary Flight Display. The annunciators on the MCP indicate the current control, lateral and vertical modes. The Primary Flight Display models the relationship between current data values, such as current altitude,

Transition: Not Active ⟶ Active

Location: Controller ↦ Vertical-Modes ↦ Intermediate-Mode ↦ Altitude

Trigger Event: MCP-Selection

Condition:

	OR			
MCP-Selection = Altitude	T	F	F	F
MCP-Selection = Glideslope	F	T	F	F
Glideslope-Position IN STATE Below-Glideslope	.	T	.	.
MCP-Selection = Minimum-Time	F	F	T	F
MCP-Selection = Minimum-Fuel	F	F	F	T

Output Action: MCP-Light-Altitude-Button

Figure 3: A definition of the transition from state NOT ACTIVE to state ACTIVE for the ALTITUDE state machine, which is an INTERMEDIATE MODE within the VERTICAL MODES of the CONTROLLER (see Figure 5). The transition can take place only when the trigger event MCP-Selection occurs and either the pilot selects the Altitude, Minimum Time or Minimum Fuel buttons on the Mode Control Panel, or the pilot selects the Glideslope button on the Mode Control Panel and the plane is below the glideslope. If the transition is taken, the Altitude button on the MCP is lit.

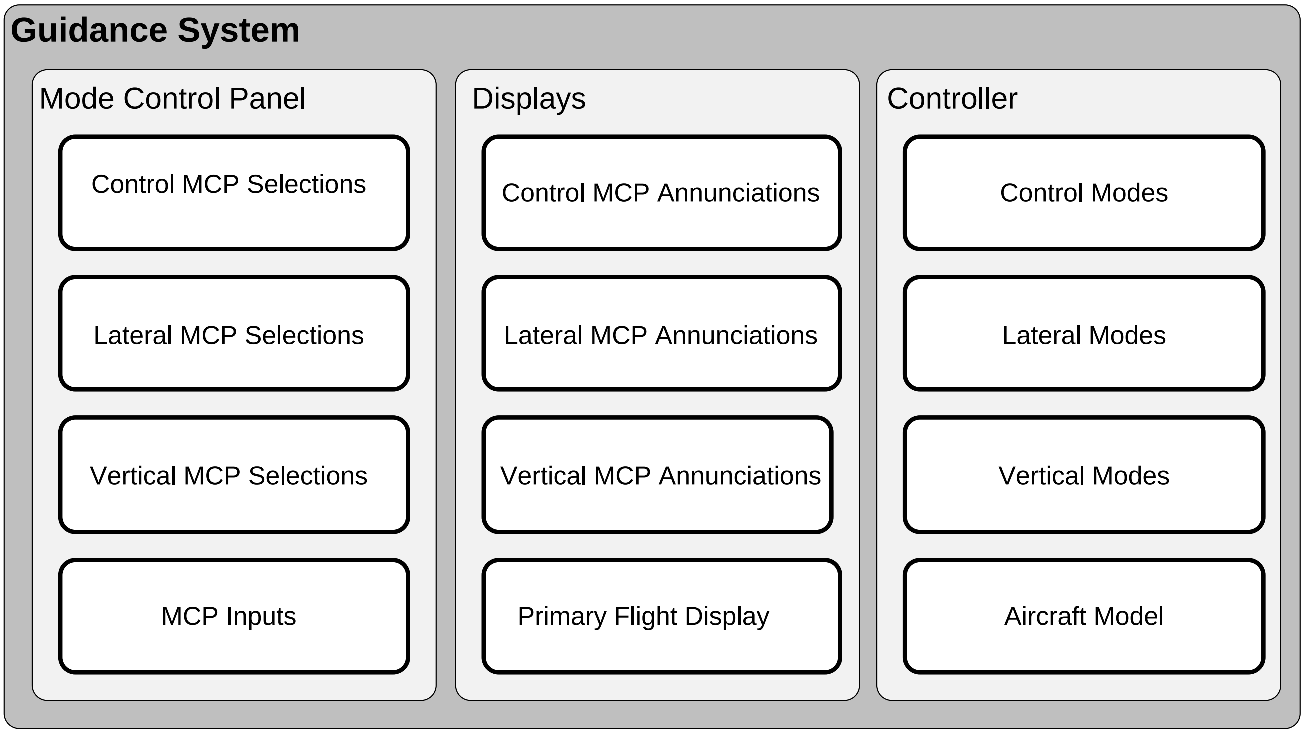

Figure 4: At a high-level of abstraction, the guidance system model consists of three parallel state machines, which themselves are composed of parallel state machines. The MODE CONTROL PANEL models the operator's input to the guidance system, which includes the control, lateral and vertical operating mode selections along with the MCP inputs such as the reference speed or reference altitude. DISPLAYS models the control, lateral and vertical modes annunciated on the MCP, and the data values displayed on the Primary Flight Display. The CONTROLLER models the logic that determines the control, lateral and vertical operating modes of the guidance system. The control logic is defined using a model of the aircraft along with the controller operating modes.

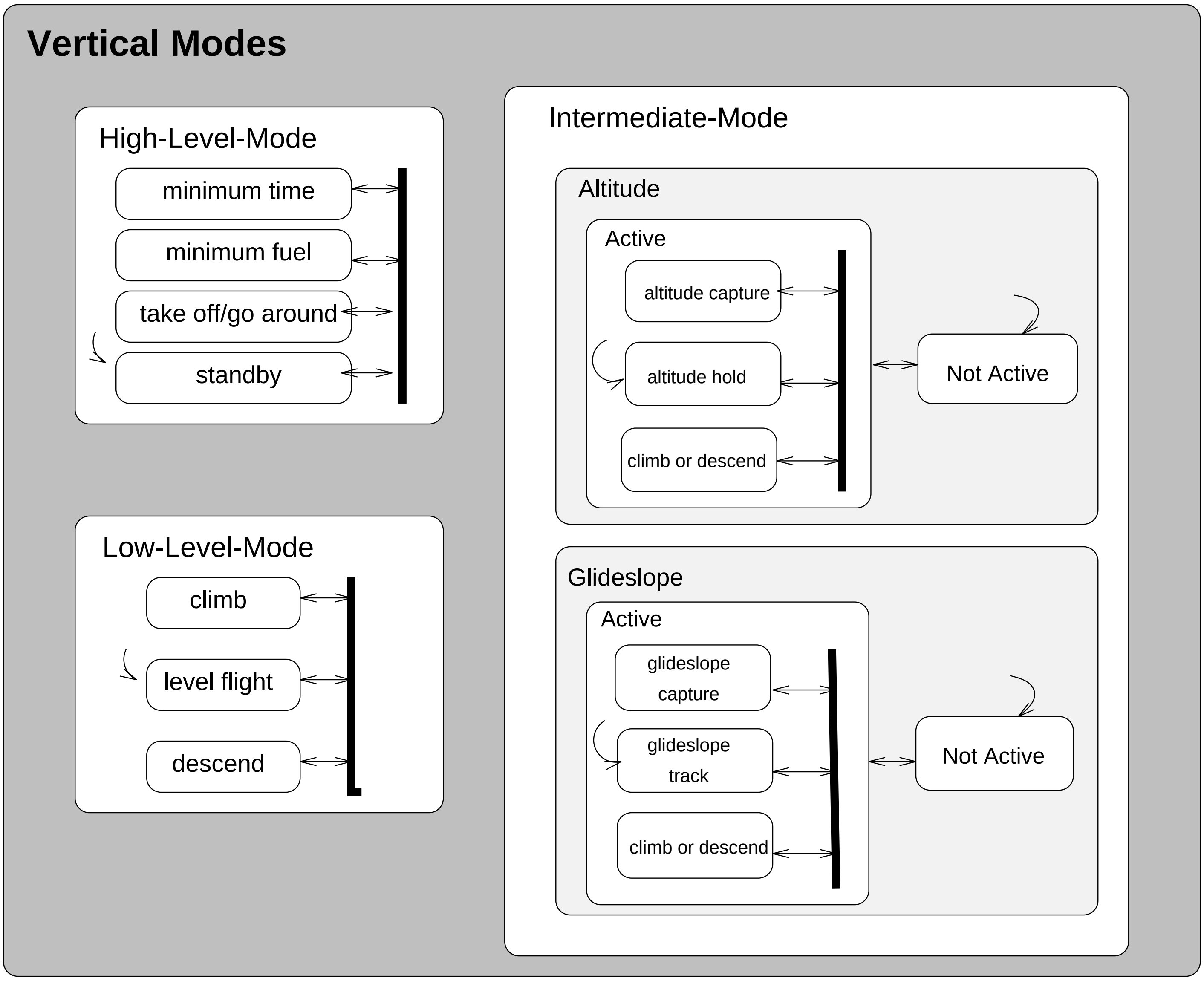

Figure 5: The VERTICAL MODES state machine of the CONTROLLER from Figure 4 further detailed. It consists of three parallel state machines: HIGH-LEVEL-MODE modeling the high-level vertical flight control modes; INTERMEDIATE-MODE modeling modes that can be initiated by either the pilot or the guidance system; and LOW-LEVEL-MODE modeling modes that can only be initiated by the guidance system.

and their desired values, such as reference altitude. Pilots use this information to monitor the behavior of the guidance system during automatic flight or to control the guidance system during manual flight.