Software Intensive Systems Safety Analysis

Alan C. Tribble & Steven P. Miller
Rockwell Collins

ABSTRACT

Two important elements in the avionics suite of modern aircraft are: the Flight Control System (FCS) and the Flight Management System (FMS). The FCS provides the capability to stabilize and control the aircraft, while the FMS is responsible for flight planning and navigation.

A clear trend in the aerospace industry is to place greater reliance on software systems, and many FCS and FMS subsystems are implemented primarily in software. For example, within the FCS is the Flight Guidance System (FGS) that generates roll and pitch guidance commands. Similarly, within the FMS is the Vertical Navigation (VNAV) function that acts like a third crew member in the cockpit, ordering mode change requests and resetting target altitude values to enable the aircraft to track the vertical flight plan.

We have developed formal, executable models of the requirements for the mode logic of a FGS and for portions of the VNAV functionality. We have also conducted a comprehensive software safety analysis on the FGS mode logic model, and are completing the analysis of the VNAV model. This analysis uses as its starting point several "traditional" safety analysis techniques such as a Functional Hazard Assessment (FHA), a Fault Tree Analysis (FTA), and a Failure Mode Effects Analysis (FMEA). However, we are also using formal methods techniques known as model checking and theorem proving to verify the presence of safety properties in the model.

This paper summarizes the (now completed) safety analysis that was performed on the FGS model, and highlights the similarities and differences with the (still on-going) safety analysis of the FMS model. In particular, we summarize progress made to date in the use of formal methods to verify the presence of the required safety properties in the models themselves.

Authors' Current Addresses:
A.C. Tribble and S.P. Miller, Rockwell Collins, 400 Collins Road, NE, Cedar Rapids, IA 52498, USA.

Manuscript received January 2, 2004; revised March 17, 2004.

0885/8985/04/ $17.00 © 2004 IEEE

INTRODUCTION

The Problem Domain

One of the challenges to investigating the feasibility and cost effectiveness of new software safety analysis techniques is developing a realistic model of a system that reflects the complexity of an actual product. The aviation domain provides several excellent candidates and the avionics system of a typical regional jet aircraft was chosen because of its safety critical nature and its inherent complexity. As Figure 1 shows, the avionics architecture is comprised of many individual systems. The gray boxes indicate those systems that are mainly non-electronic, (i.e., little software) in nature. The white boxes indicate those systems that are electronic, and often have high software content like the Flight Management System (FMS) and the Flight Control System (FCS). The FMS is decomposed into discrete and continuous elements called the mode logic, the control laws, and the flight plan. The FMS mode logic is a set of discrete algorithms that determine when the FGS should change modes of operation. The control laws are continuous trajectory calculations that compare the measured state of the aircraft (position, speed, attitude, altitude), to the desired state and generate guidance commands to minimize the difference between the two. Finally, the flight plan is defined by the flight crew and specifies the desired trajectory of the aircraft, based on adherence to constraints on altitude and position. The FCS also contains mode logic and flight control laws, known as the Flight Guidance System (FGS), in addition to the Flight Director (FD), Auto-Pilot (AP), and Auto-Throttle (AT). Other elements, not shown, may include a yaw damper and auto-trim.

The FGS is a software function that generates roll and pitch values used to control the aircraft, and was selected as the example for our analysis. The FGS is decomposed into discrete and continuous elements called the mode logic and the flight control laws, respectively. The flight control laws compare the measured state of the aircraft (position, speed, attitude, altitude, to the desired state and generate guidance commands to minimize the difference between the two. The mode logic selects the appropriate flight control laws for use anytime the system is active. In contrast, the FMS is responsible for a more diverse set of functions ranging from flight planning to navigation.

Accident Model

Although thorough knowledge of the nature of accidents is not necessary to appreciate the value of our results, a high level understanding is helpful to see how this same approach could

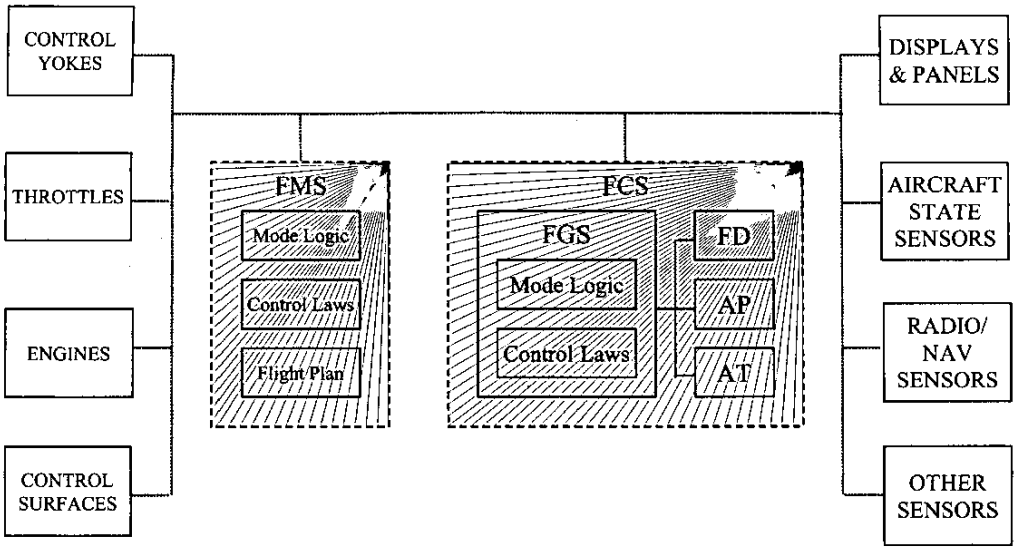

Fig. 1. The high level architecture of an avionics system

be applied in a larger context. Underlying our analysis is an assumption about the nature of accidents as shown in Figure 2. The definitions used in this accident model are in general agreement with IEEE standards [1, 2]. In brief, an error may be manifested as a fault, a fault may result in a failure, a failure may place the system in a hazardous condition, and a hazardous condition may result in an accident [1].

Our safety analysis therefore focuses on defining the hazards, failures, faults, and errors that could lead to accidents. As shown later, our analysis will use a combination of standard techniques, (e.g., Fault Tree Analysis and Failure Modes, Effects, and Criticality Analysis), in combination with non-traditional, yet very powerful, formal methods techniques.

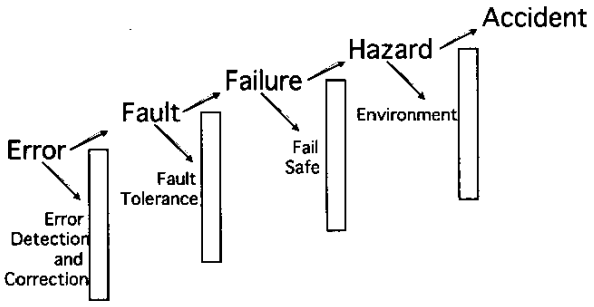

Fig. 2. The sequence of events leading to an accident

SOFTWARE SAFETY ANALYSIS

Specifying the Requirements

A specification of the FGS mode logic, and portions of the VNAV function, has been generated in a formal language, the Requirements State Machine Language without Events (RSML$^-$). RSML$^-$ is a synchronous language developed for specifying the behavior of process control systems [3]. RSML$^-$ runs in the "Nimbus" environment developed by the Critical Systems Research Group at the University of Minnesota. The environment provides a framework for the development of software for safety critical systems, including simulation and visualization. In particular, the Nimbus environment includes a graphical user interface for the simulation engine. An advantage of RSML$^-$ is that it is executable. That is, a user may provide inputs and watch how the state machines respond. This makes it ideal for use in a model-based development environment where the requirements themselves can be verified early in the design and development process, while the cost of correcting them is still low. Another important advantage is that RSML$^-$ possesses a precise formal semantics so that the models can be formally analyzed.

The predecessor to RSML$^-$, RSML was heavily influenced by Statecharts and uses a similar notion of explicit event propagation. RSML was used to specify the Traffic Collision Avoidance System II (TCAS-II) and the RSML model was ultimately adopted by the FAA as the official specification for TCAS-II. As its name implies, RSML$^-$ eliminates the use of events and its semantics have been fully formally defined. RSML$^-$ is in most respects similar to SpecTRM-RL, developed by the Safeware Engineering Corporation, but has a slightly different syntax and underlying philosophy.

DEFINING THE SAFETY PROPERTIES

The next step in the safety analysis process is to formally define those properties of the software associated with safety. Safety properties were generated via a Bi-Directional Analysis

[1] It should be noted that not every accident will be initiated by an error. The initiating event may be a fault, as in the case of a single event upset (SEU) in electronic devices, or a failure, due to the wear-out of hardware devices.

(BDA) technique [4, 5]. The starting point for the BDA is the list of hazards. Top-down analysis is then used to trace the hazards down to the related errors. To close the loop, an independent bottom-up analysis is used to trace the errors back up to hazards.

Functional Hazard Assessment (FHA)

Safety is a system level problem and aviation safety standards ARP 4754 and ARP 4761 specify that safety analysis be performed both at the aircraft level and at the system level [6, 7]. The aircraft level hazards are generally few, such as loss of control. If the loss of control hazard is examined, it can be found that failures in a number of systems, (e.g., hydraulic lines, control yokes, flight control surfaces), could give rise to it. These hazards will derive from functional failures and are defined in a Functional Hazard Assessment (FHA).

We started with the functional requirements for the system in question. Examining the consequences of the system failing to provide this functionality identified the hazards associated with the function. Each of these hazards was then assigned a level of criticality in accordance with DO-178B and MIL-STD-882 [8, 9].

We have completed the FHA for both the FGS mode logic and VNAV models. The FGS mode logic analysis is complete, [10, 11], while the VNAV analysis is still being refined [12]. As such, the majority of our discussion will focus on describing the results from the analysis of the FGS mode logic model. After completing this analysis, we identified four Level C (Major) hazards. Because Level C is the most critical hazard, the FGS is considered a Level C system. (In comparison, VNAV is usually considered a Level B system.) The FHA for the Level C hazards is shown in Table 1.

Fault Tree Analysis (FTA)

Fault Tree Analysis (FTA) is a top-down analysis technique used to identify the contributing elements (errors/faults/ failures) that could precipitate the system level hazards identified [2, 13]. FTA is a feed-back technique in that one starts with the system level hazards and attempts to work backward by identifying all possible causes of the hazards. Although the name implies that the technique is limited only to "faults," it should be emphasized that FTA is a general, visual technique used to trace higher level events (such as hazards) down to their contributing events. These contributing events could be failures, or errors, in addition to faults.

In an actual aircraft program, the FTA would start with the system level hazards; for example, Loss of Control, and include all aircraft systems that could potentially contribute to such a hazard. For our purposes, the FTA will start with the hazards identified in the FHA. For example, the FTA associated with the hazard "Incorrect Guidance" is shown in Figure 3. Note that because safety is a system level property, the FTA must include elements that input information to the FGS, such as the Flight Control Laws (FCL), and elements that the FGS outputs information to, such as the AP or FD. By performing a FTA on each of the four hazards listed in Table 1, we obtained a listing of twenty-three (23) possible events that could contribute to hazardous conditions. Of these, eight (8) were relevant to the FGS mode logic.

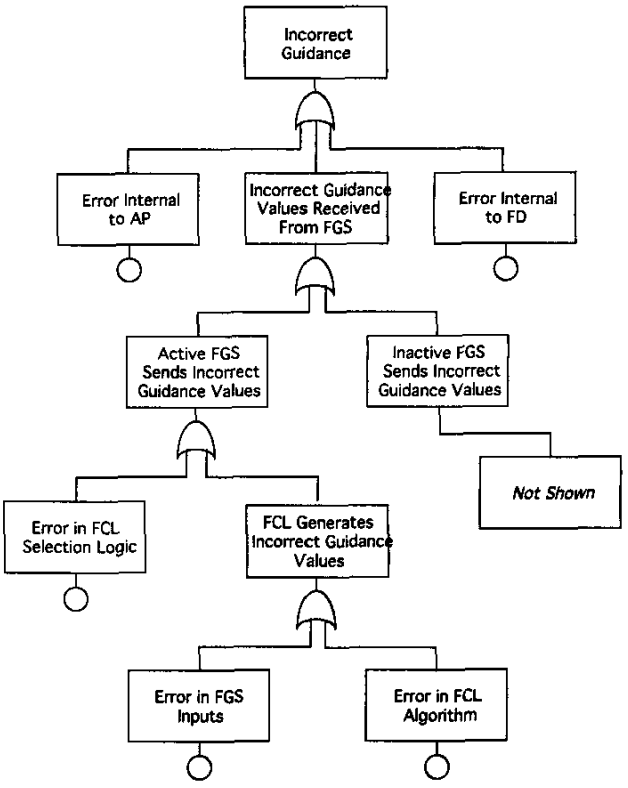

Fig. 3. A top level view of the fault tree analysis for the hazard – incorrect guidance

Failure Mode Effects Analysis (FMEA)

To check the results from the top-down FTA, we conducted a bottom-up Failure Mode Effects Analysis (FMEA). FMEA is a feed-forward technique in that the starting points for the analysis are possible errors which are then traced forward to see if they have any impact on system safety (i.e., if they lead to potential hazards) [2, 13]. As with the FTA, the term FMEA should not imply that the results are limited to "failures." FMEA is a general analysis method that flows errors (or faults or failures) forward to hazardous conditions.

The output of a FMEA is a tabular presentation that lists: 1) failure mode (error); 2) effects (hazard); and 3) analysis (interpretation). The starting point for the FMEA was the list of errors derived from Table 1. Many of these errors are associated with hardware or are dependent on malfunctions in other systems and were considered out of scope for this software safety analysis. In this instance, the FMEA did not uncover any new failures but rather confirmed the results generated by the FTA. This is one of the advantages of BDA; tracing through the accident process in both directions gives higher confidence in the final results.

Table 1. The Functional Hazard Assessment Identified Four Level C Hazards

Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Comment
Incorrect Guidance	Approach	Gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	No Difference to the AP Between Loss of Guidance and Incorrect Guidance.
Incorrect Mode Indication	Approach	Gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	Assumes guidance values are correct.
Incorrect Indication of Flight Guidance Transfer State.	All	Incorrect "Pilot Flying" side indicated. Possible gradual departure from references until detected by flight crew during check of primary flight data resulting in manual disconnect and manual flying.	Departure from references occurs only if pilot flying and pilot not flying have selected different navigation sources.
Incorrect AP Engagement Indication	Approach	If engaged, engagement noticed by resistance to control column / wheel inputs. If disengaged, departure from references noticed during check of primary flight data. Result is manual disconnect and manual flying.	Assumes AP disconnect remains operational

Safety Properties

The eight categories of software errors that were in scope were further examined, to identify the specific properties of the software that could produce the higher level events. For example, in the final model it was seen that there were forty-one (41) separate functional properties associated with the "Error in Annunciation Logic" category. Nine (9) of these "functional" requirements are truly "safety" properties in that violating them may place the system in a hazardous condition. Special emphasis should be placed on verifying these properties that relate to safety.

As shown in Table 2, the FGS mode logic model contained 293 distinct functional requirements, or functional properties. The safety analysis showed that 155 of these "functional" properties were truly "safety" properties that could result in one of the four Level C (Major) hazards identified. We believe that the analysis to this point is unique in that this level of detail is not usually conducted on Level C (or Level B) systems. However, we have also taken our analysis to an even higher level by expanding it to include Formal Methods techniques as discussed in the next section.

FORMAL METHODS APPROACH

The term *Formal Methods* refers to a variety of mathematical modeling techniques applicable to computer system (software and hardware) design. In much the same way that aeronautical engineers make use of computational fluid dynamics (CFD) to predict how a particular airframe design will behave in flight, computer scientists may use formal methods to predict the behavior of software or hardware. Two of the most popular formal verification tools are: model checkers and theorem provers [14, 15].

Theorem proving is a technique where both the system and its desired properties are expressed as formulas in mathematical logic. Proving a theorem is simply the process of verifying the existence of a mathematical property from the specifications of the system. Although in principle all proofs could be done manually, it is more effective to use machine based theorem provers to tackle larger, more realistic problems, such as the FGS mode logic or VNAV.

Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in the model. Checking a model is the process of performing an exhaustive state space search, which is guaranteed to terminate if the model is finite, to look for examples that do not meet the property desired. If a counter-example is found, it is known that the property does not hold.

The use of formal methods in assessing software safety involves four steps. First, the software itself must be specified in a formal language. Second, the safety properties must also be defined formally. Third, both the specification and the

Table 2. A Total of 142 Safety Properties were Identified for the Mode Logic

			# of Properties Contributing to a Level C (Major) Hazard			
		# of Properties	Incorrect Guidance	Incorrect Mode Indication	Incorrect Indication of Flight Transfer State	Incorrect AP Engagement Indication
Safety Property Category	Error in Annunciation. Logic	41	-	9	-	-
	Error in FD Selection Logic	13	-	-	-	-
	Error in Pilot Flying Transfer Logic	8	-	-	4	-
	Error in Independent / Active Logic	5	5	5	-	-
	Error in AP Engagement Logic	10	-	-	-	4
	Error in Mode Selection Logic	166	-	104	-	-
	Error in Synchronization Logic	50	-	23	-	-
	Total # of Properties	**293**	***5***	***141***	***4***	***10***

safety properties must be translated into formal methods tools capable of performing the analysis. Finally, the analysis itself is conducted. An overview of each of these steps, as they have been applied to our problem domain, is provided in the next section.

Translating to Analysis Tools

It is certainly possible to define the requirements and safety properties in the same language initially, but for real projects this will rarely be the case. The requirements model will probably be developed in tools such as RSML^{-e}, Matlab or SCADE. Safety properties will probably be defined in English prose. Even if the properties were defined in the requirements modeling tool, the modeling tool itself would probably lack the analysis capability and would require translation to a theorem prover or model checker to enable the actual analysis. The RSML^{-e} tool has automated translators to both the NuSMV model checker and the PVS theorem prover [16, 17].

We were extremely pleased to see that the FGS mode logic model was completely analyzable with a model checker. That is, all 293 functional (142 safety) properties were verified by the NuSMV model checker in about two hours. This confirms that model checking technology is advanced enough for use on models such as the FGS mode logic, which are composed of primarily discrete logic. Analysis of the FMS VNAV model has not yet been completed. The FMS VNAV model utilizes integers and reals, making it a much larger state space. For example, most altitude values are specified in feet, with possible values ranging from –1,300.00 to + 65,000.00. Truncating the accuracy at two decimal points would result in 6,630,000 possible numerical values for all altitude variables. In theory, the combined state space for two altitude variables, (such as the Pre-Select Altitude and the Flight Plan Target Altitude), would be the product of the state space for either variable, or more than 10^{12}. This is the equivalent state space of over 40 independent Boolean variables. We are investigating different approaches that may enable the model checker to solve the FMS VNAV problem [12], but in the meantime we are proceeding with the use of the PVS theorem prover.

In comparison to model checkers, which are easy to use but limited by the state space explosion problem, two key advantages of theorem provers are largely unaffected by the size of the state space. Unfortunately, theorem provers are more difficult to use and require much more manual oversight than model checkers, which are very automated. A theorem prover applies the rules of logic to reason about the system in question. Theorem proving is the process of:

- Specifying the system, assumptions, necessary background theories, and desired properties, as formulas in a single logic, then;
- Proving that the property is modeled by the system + assumptions + background.

Using the processing speed and memory of modern computing systems makes the theorem prover capable of rigorously dealing with much larger and more complex problems than the human mind can, but the reasoning process is – for the most part – still guided by the human user. For example, the theorem prover may try to prove a lemma by induction, only to generate different subgoals that must be proven individually before the proof can be completed. The user must provide guidance on what rules of inference should be used, and in what order they should be applied, for the theorem prover to complete the task.

To date, we have succeeded in proving many properties for both the FGS model logic and VNAV models. The results confirm the advantage of using theorem provers for problems of this nature, but also highlight the difficulties in transitioning the results from the laboratory to a production environment. We are confident that the theorem prover is capable of discharging all properties (provided we have indeed constructed the models correctly) and we are continuing to investigate methods that will speed the analysis so that other programs can benefit from our results.

SUMMARY AND CONCLUSIONS

We have constructed formal, executable models of two complex, embedded software systems – the mode logic of a Flight Guidance System and the Vertical Navigation function. Having the ability to simulate the models helps to verify the correctness and completeness of the requirements and is also a starting point for further model-based development. In order to identify the properties of the software that are related to safety, we then conducted a thorough software safety analysis using standard techniques such as Functional Hazard Assessment (FHA), Fault Tree Analysis (FTA), and Failure Mode Effects Analysis (FMEA). To verify that the model did indeed contain the safety properties required, we then conducted an extended software safety analysis on the design requirements using two formal methods techniques: model checking and theorem proving. In particular, we have used a model checker to show that almost 300 functional (and safety) properties associated with the FGS mode logic are mathematically verifiable properties of the model. We have also used a theorem prover to verify a smaller number of properties for both the FGS mode logic and VNAV models.

We have found the use of formal requirements modeling, in conjunction with Bi-Directional Analysis and formal methods analysis techniques, to be a cost-effective and flexible approach to the issue of software safety. In particular, we were impressed with the power of model checking as applied to this example. This lends credence to the belief that such approaches may become an integral part of future model-based development efforts [18].

ACKNOWLEDGEMENTS

This work was supported, in part, by the NASA Aviation Safety Program and the NASA Langley Research Center under contract NCC-01-001. This paper is a summary of reports presented at the 2002 and 2003 Digital Avionics Systems Conferences (DASC) [10, 12].

REFERENCES

[1] IEEE Std. 610.12-1990,
Standard Glossary of Software Engineering Terminology.

[2] Herrmann, D.S.,
Software Safety and Reliability,
(Los Alamitos, CA: IEEE Computer Society, 1999).

[3] Whalen, M.W.,
A Formal Semantics for RSML, Masters Thesis,
University of Minnesota, April 2000.

[4] Lutz, R. and R. Woodhouse, 1997,
Requirements Analysis Using Forward and Backward Search,
Annals of Software Engineering, Vol. 3, pp. 459-475, 1997.

[5] Lutz, R. and R. Woodhouse, 1999,
Bi-Directional Analysis for Certification of Safety-Critical Software,
Proc. of First International Software Assurance Certification Conference, 1999.

[6] ARP 4754,
Certification Considerations for Highly-Integrated or Complex Aircraft Systems, SAE International, November 1996.

[7] ARP 4761,
Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,
SAE International, December 1996.

[8] RTCA DO-178B,
Software Considerations in Airborne Systems and Equipment Certification, 1 December 1992.

[9] MIL STD 882C,
System Safety Program Requirements, 19 January 1993.

[10] Tribble, A.C., S.P. Miller and D.L. Lempia, 29-31 October 2002,
Software Safety Analysis of a Flight Guidance System,
Proceedings of the 21st Digital Avionics Systems Conference, Irvine, CA, 29 - 31 October 2002.

[11] Tribble, A.C., S.P. Miller and D.L. Lempia,
Software Safety Analysis of a Flight Guidance System,
NASA Contractors Report, [in press].

[12] Tribble, A.C. and S.P. Miller, 12-16 October 2003,
Software Safety Analysis of a Flight Management System Vertical Navigation Function – A Status Report,
Proceedings of the 22nd Digital Avionics Systems Conference, Indianapolis, IN, 12 – 16 October 2003.

[13] *System Safety Analysis Handbook*, 2nd Ed.,
System Safety Society, July 1997.

[14] Clarke, E.O., O. Grumberg and D. Peled, 2000,
Model Checking, (Cambridge, MA: MIT Press, 2000).

[15] Butler, R.W., September 1993,
An Elementary Tutorial on Formal Specification and Verification Using PVS, NASA TM 108991, September 1993.

[16] NuSMV: A New Symbolic Model Checker,
available at http://nusmv.irst.itc.it/.

[17] PVS: Prototype Verification System,
available at http://www.csl.sri.com/projects/pvs/.

[18] Tribble, A.C., June - July 2002,
Software Safety, *IEEE Software*, pp. 84 - 85, June – July 2002.