DOT/FAA/TC-19/47

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Identify Safety Issues in Integration of Complex Digital Systems

March 2020

Final Report

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

1. Report No. DOT/FAA/TC-19/47	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle IDENTIFY SAFETY ISSUES IN INTEGRATION OF COMPLEX DIGITAL SYSTEMS		5. Report Date March 2020 6. Performing Organization Code
7. Author(s) Chris Wilkinson, Brendan Hall, Kevin Driscoll (Honeywell) Nancy Leveson (MIT) Paul Snyder (UND) Frank McCormick (CSI)		8. Performing Organization Report No.
9. Performing Organization Name and Address Honeywell Aerospace 7000 Columbia Gateway Drive Columbia, MD 21046		10. Work Unit No. (TRAIS) 11. Contract or Grant No. DTFACT-15-C-00024
12. Sponsoring Agency Name and Address FAA, William J Hughes Technical Center Atlantic City International Airport, NJ 08405		13. Type of Report and Period Covered Final Report 14. Sponsoring Agency Code AIR-130

15. Supplementary Notes

The FAA William J. Hughes Technical Center Aviation Research Division COR was John Zvanya.

16. Abstract

This report details investigations into the underlying causes of, and methods to reduce, the likelihood of integration-related incidents and accidents. A summary is provided of some accidents of this type and what lessons were learned. The use of formal methods as a tool to improve the correctness and completeness of functional and safety requirements are then explored, with one case study presented and a survey of available tools. An outline is then presented of a proposed development process incorporating state-of-the-art safety assessment, requirements specification, and validation methods. Finally, the use and misuse of the Technical Standard Order Approval process are discussed and the use of Technical Standard Order (TSO) products incorporated into larger integrated systems, examining an accident whose proximate cause was a TSO product. To summarize, some tentative conclusions and recommendations for improvements to current certification guidance are discussed.

17. Key Words Complex systems, Integration, Safety, Accidents and incident, Formal methods	18. Distribution Statement This document is available to the U.S. public through the National Technical Information Service (NTIS), Springfield, Virginia 22161. This document is also available from the Federal Aviation Administration William J. Hughes Technical Center at actlibrary.tc.faa.gov.

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 141	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

ACKNOWLEDGEMENTS

The authors thank the following individuals for their thoughtful suggestions and comments:

Manuel Rios (FAA) and Srini Mandalapu (FAA).

TABLE OF CONTENTS

LIST OF FIGURES

ACRONYMS AND ABBREVIATIONS

AADL	Architectural Analysis and Design Language
ACO	Aircraft Certification Office
AD	Airworthiness directive
ADIRU	Air data inertial reference unit
AGREE	Assume Guarantee Reasoning Environment
AOA	Angle of attack
ATSB	Australian Transportation Safety Board
BAA	Broad Agency Announcement
BLESS	Behavioral Language for Embedded System Software
BSCU	Brake and steering control unit
CAS	Computed airspeed
CAST	Causal analysis based on system theory
CBD	Contract-based design
CBSA	Contract-based safety analysis
CLP	Constraint Logic Programming
COM/MON	Command/monitor
CRC	Cyclic redundancy checks
DAL	Design assurance level
DFM	Dynamic Flowgraph Methodology
EARS	Easy Approach to Requirements Syntax
EASA	European Aviation Safety Agency
ECAM	Electronic centralized aircraft monitor
EICAS	Engine-Indicating and Crew-Alerting System
ESA	European Space Agency
FBK	Fondazione Bruno Kessler
FCMC	Fuel control and monitoring computer
FCU	Flap control unit
FDIR	Fault detection, isolation, and recovery
FDR	Flight data recorder
FEI	Fault extension file
FMEA	Failure Mode and Effect Analysis
FMS	Flight management system
FTA	Fault tree analysis
GSN	Goal Structured Notation
GUI	Graphical User Interface
HAZOP	Hazard and Operability Analysis
HiP-Hops	Hierarchically Performed Hazard Origin and Propagation Studies
HMI	Hazardously misleading information
IC	Integrated circuit
IDE	Integrated Development Environment
IMA	Integrated Modular Avionics
LE	Leading edge
LMP	Logical model processing
LOC	Loss of control
LRU	Line Replaceable Unit

LTL	Linear temporal logic
MBSA	Model-based safety analysis
MBSE	Model-based system engineering
MEL	Minimum equipment list
MOPS	Minimum Operational Performance Specification
NTSB	National Transportation Safety Board
OCRA	Othello Contracts Refinement Analysis
OSS	Othello System Specification
PDU	Pilot display unit
PFD	Primary flight display
PMA	Parts manufacturer approval
PRA	Probabilistic risk analysis
RCCA	Root cause corrective action
RCP	Rich client platform
RTL	Rudder travel limiter
SAARU	Standby Attitude and Air data Reference Unit
SAE	Society of Aerospace Engineers
SMS	Safety Management System
SMT	Satisfiability Modulo Theory
SOP	Standard operating procedure
SpecTRM	Specification Tools and Requirements Methodology
SRA	Safety risk assessment
STAMP	System Theoretic Accident Model and Process
STC	Supplemental type certificate
STPA	System Theoretic Process Analysis
SysML	System modeling language
TC	Type certification
TCAS	Traffic Collision and Alerting System
TFPG	Timed failure propagations graphs
TLE	Top-level event
TR	Thrust reverser
TSO	Technical Standard Order
TSOA	Technical Standard Order Authorization
UML	Unified modeling language
V&V	Verification and validation
VM	Virtual Machine
VR	Rotation speed

EXECUTIVE SUMMARY

This is the final report for FAA Broad Agency Announcement (BAA) TCBAA-15-00001 entitled "Identify Safety Issues in Integration of Complex Digital Systems." The work performed by Honeywell, the Massachusetts Institute of Technology, the University of North Dakota, and Certification Services Inc. identified safety issues related to verification and validation during the design of highly integrated complex digital systems, including gaps in the design/development assurance processes, Technical Standard Order authorization of articles, and the impact that changes within one or more interconnected systems can have on another. This BAA defined a successful outcome as having a direct benefit to the FAA and industry in terms of:

- Facilitating early validation of requirements used in integration of complex digital systems.
- Ensuring completeness and correctness of guideline standards by identifying missing or incorrect system-level requirements.
- Enhancing understanding of the failure and failure effects from the functional integration of digital systems and potential impact on aircraft safety from the failure of similar systems.
- Identifying issues when independently approved components are integrated into complex digital systems.

This work began with a survey of aircraft accidents (to identify common themes in complex system failures) and a survey of technologies and tools related to system- and safety-related engineering (to establish the state-of-the art in model-based safety analysis and related formal method technologies). Following these surveys, effort was focused on a case study that applied selected formal methods to a wheel brake system. Conclusions from that work are included herein. Drawing on lessons learned from the surveys and the case study, investigations were made of emerging technologies that may help mitigate the risks of complex system development. A vision for a knowledge-based approach was developed, and several experiments were completed. This effort also investigated issues related to change management, drawing from the principles and rationale of the Safety Management System philosophy to better address the challenges and gaps related to current certification guidance.

From this work, the concept of "principled design and architecture refinement" was developed. This can add a "step 3" to the System Theoretic Process Analysis process, of which steps 1 and 2 focus on the early stages of the design process. This concept shows great promise in not only providing the benefits in the list above, but also in addressing the lack of sufficient expertise in highly dependable design among current and future system designers. Tools and processes must be created to implement this principled design and architecture refinement process to avoid the repetition of known architectural design errors, which will be exacerbated by the ever-increasing complexity of digital systems.

1. INTRODUCTION

This is the final report on a research program sponsored by the FAA commissioned Broad Agency Announcement (BAA) TCBAA-15-00001, entitled "Identify Safety Issues in Integration of Complex Digital Systems." The tasks performed included:

Identify safety issues related to verification and validation (V&V) during the design of highly integrated complex digital systems.

Identify gaps in the design/development assurance processes used to verify and validate integrated complex digital systems, and identify any potential failures and failure effects resulting from such complex systems integration.

Identify safety issues related to Technical Standard Order (TSO) authorization of articles used in complex digital systems.

Identify change impact analysis issues associated with integration of complex digital systems, and identify any potential safety implications due to any change(s) within one or more functionally interconnected systems.

- The intention of this work is to provide a successful outcome for this BAA that will have a direct benefit to the FAA and industry in terms of:

  a) Facilitating early validation and verification of requirements used in integration of complex digital systems.
  b) Ensuring completeness and correctness of the guideline standards by identifying the missing or incorrect system-level requirements, which can be attributed to assurance of complex digital systems.
  c) Enhancing understanding of the failure and failure effects from the functional integration of digital systems and the potential impact on aircraft safety from the failure of similar systems.
  d) Identifying issues when independently approved components are integrated into complex digital systems.

The work described herein was split into two phases. The first portion (approximately 70% of the total research effort) of this research effort was largely conducted by the initial principal investigator, Chris Wilkinson. To prevent data loss, the content of the interim report has largely been retained within this document, in sections 2–8. This work comprises a survey of aircraft accidents (with a goal of identifying common themes in complex system failures) and a survey of technologies and tools related to system and safety-related engineering (to establish the state-of-the art in model-based safety analysis and related formal method technologies). Following these surveys, significant effort was then focused on a case study that applied selected formal methods to a wheel-brake system. Wilkinson's interim conclusions from the work that he performed are included.

Following Wilkinson's retirement, the remaining Honeywell effort (approximately 30%) was focused on reintegrating the broader perspectives of the wider research team. Drawing from the lessons learned from the first effort, this second phase investigates emerging technologies that may help mitigate the risks of complex system development. A vision for a knowledge-based approach is presented, as are several experiments investigating and exploring the building blocks and enabling technologies to support such a vision. The latter effort also investigated issues related to change management and draws from the principles and rationale of the Safety Management System (SMS) philosophy to better address the challenges and gaps related to current certification guidance.

The research team executing these tasks is comprised of members from Honeywell Aerospace Advanced Technology, the Massachusetts Institute of Technology, Certification Services Inc., the University of North Dakota, and the Honeywell product integrity organization.

2. PROBLEM DEFINITION

There are many causes for accidents and incidents. This work focuses on those that can loosely be described as integration related (i.e., some fault, failure, or design error had a ripple effect through other connected systems, ultimately manifesting as unsafe behavior of the aircraft). Safety issues in integration have some general characteristics. There are two or more communicating components for which components are hardware, software, or subsystems. By this definition, a Line Replaceable Unit (LRU), Line Replaceable Module, Circuit Card Assembly, partition, and thread are all components. As integrated circuit (IC) technology becomes more powerful and complex, the majority of what had been an aircraft subsystem could now reside on a single IC, becoming a significant component on its own. A component can provide unexpected, unintended, erroneous, or no output. These misbehaviors can be due to a component's processing of erroneous external signal inputs or because of an internal fault or failure. A failure state may be entered by the effects of external environmental inputs, such as temperature, radiation, electromagnetic interference, and power failure. The definition to the integration of subsystems into larger ones is not restricted, but also includes the same types of issues arising during the integration of the hardware and software components of a moderately complex subsystem that may have been created within one company under one set of design rules.

A second type of integration issue arises if there are differing views of the semantics of data that cross integration boundaries. A lack of a well-defined and consistent understanding among stakeholders of what constitutes a single truth can lead to a major system or a subsystem misinterpreting the meaning of a data item, or to different subsystems interpreting the meaning of the same data item differently. The latter case is often difficult to detect.

A typical feature of integration-related incidents is that components downstream of an error source react to source errors as they were designed to do (i.e., the downstream components meet their given design requirements, functioning as designed). This circumstance implies a missing root cause or incomplete requirements. In such circumstances, verification standards, such as DO178C [1] and DO-254 [2], do not apply. The resulting behavior can be unsafe at the aircraft level.

3. SURVEY OF ACCIDENTS AND INCIDENTS

To seed the research, the Honeywell team conducted a review of previous accidents by surveying public investigation reports and private knowledge. Examples have been selected that exhibit the characteristics described in the problem definition to identify areas in which improved processes and developments methodologies may add value.

3.1 PUBLICLY AVAILABLE ACCIDENT REPORTS

3.1.1 Qantas A330, Near Learmonth Australia 2008

A faulty air data inertial reference unit (ADIRU) provided bad, spiky angle of attack (AOA) data to the flight control system, causing an abrupt pitch down and passenger injury. Fifty-three people were taken to the hospital, 12 of whom were seriously injured. This fault was announced to the crew via electronic centralized aircraft monitor (ECAM), and the crew managed to recover manual control and made an emergency landing at Learmonth [3]. This incident resulted in an emergency European Aviation Safety Agency (EASA) airworthiness directive (AD) in January 2009. The Australian Transportation Safety Board (ATSB) report remains in active status with an ongoing investigation. This incident may also be regarded as symptomatic of a system-design error (the flight-control system did not adequately check for reasonableness of the incoming AOA data or monitor the ADIRU health status), which led to using the bad data.

ADIRUs have been implicated in several other similar A330 incidents. The aircraft involved in the first three listed below were fitted with the same model of Northrop-Grumman ADIRU. These three incidents are also described in the same report (pages 31–35).

- Qantas QF68 (VH-QPA), 12 September 2006
- Jetstar JQ7 (VH-EBC), 7 February 2008
- Qantas QF71 (VH-QPG), 27 December 2008
- Air France AF447 (F-GZCP), 1 June 2009 [4]

The ADIRU incorrectly handled air data signal spikes, and V&V had not considered the effect of frequent spikes. This signal processing and its effects are shown in figure 1. The emergency EASA AD in January 2009 required crews to turn off the ADIRU if faulty data are detected. The accident report recommended eight safety actions.

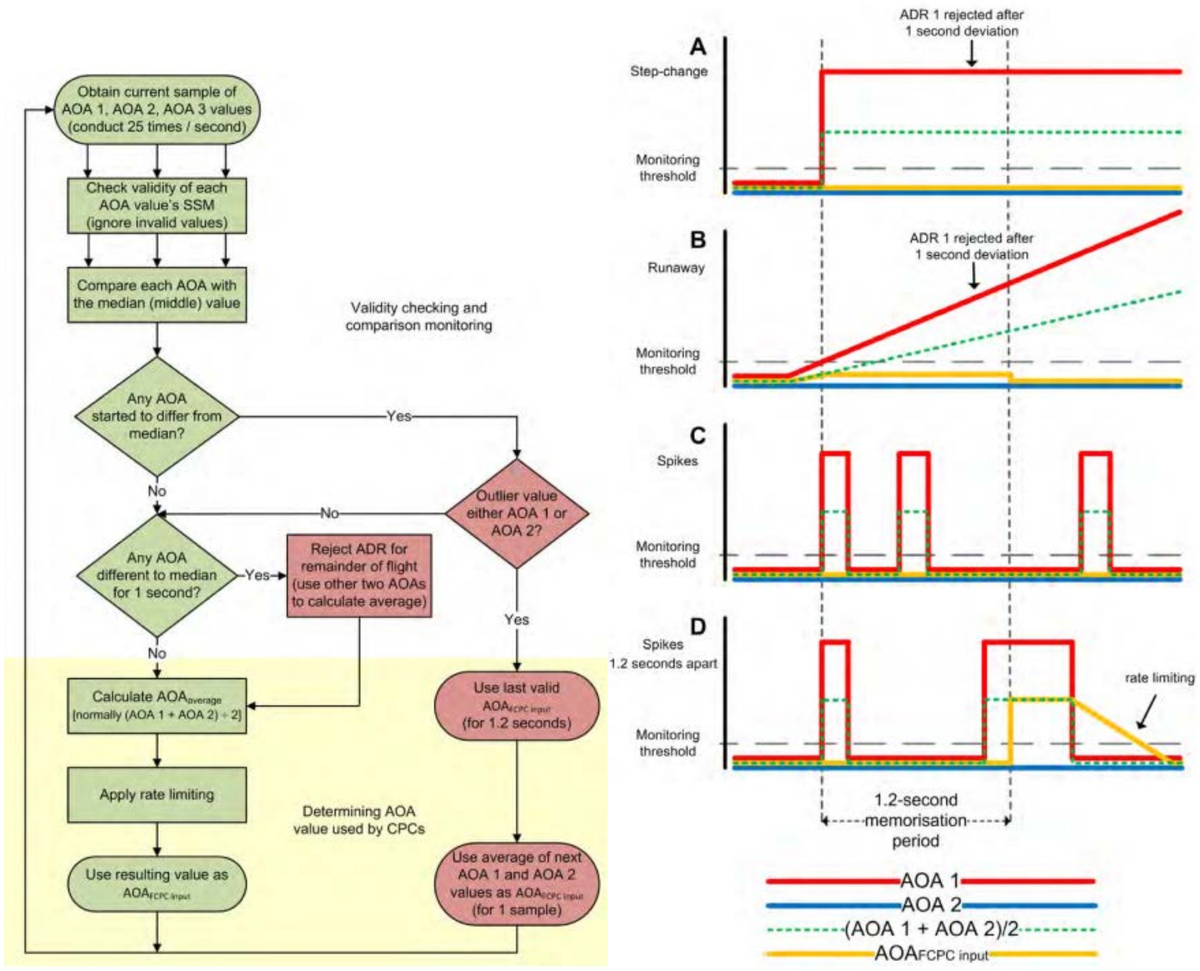

Obtain current sample of AOA 1, AOA 2, AOA 3 values (conduct 25 times / second)
Check validity of each AOA value's SSM (ignore invalid values)
Compare each AOA with the median (middle) value
Validity checking and comparison monitoring
Any AOA started to differ from median?
Yes
No
Outlier value either AOA 1 or AOA 2?
Any AOA different to median for 1 second?
Reject ADR for remainder of flight (use other two AOAs to calculate average)
Calculate AOAaverage [normally (AOA 1 + AOA 2) ÷ 2]
Use last valid AOAFCPC input (for 1.2 seconds)
Apply rate limiting
Determining AOA value used by CPCs
Use resulting value as AOAFCPC input
Use average of next AOA 1 and AOA 2 values as AOAFCPC input (for 1 sample)
A Step-change
ADR 1 rejected after 1 second deviation
Monitoring threshold
B Runaway
C Spikes
D Spikes 1.2 seconds apart
rate limiting
1.2-second memorisation period
AOA 1
AOA 2
(AOA 1 + AOA 2)/2
AOAFCPC input

Figure 1. Signal processing and its effects

3.1.2 Malaysian B777, Perth 2005

At approximately 1703 Western Standard Time, on 1 August 2005, a Boeing Company 777-200 aircraft, registered 9M-MRG, was operating on a scheduled international passenger service from Perth to Kuala Lumpur, Malaysia [5]. The crew reported that, during climb out, they observed a LOW AIRSPEED advisory on the Engine-Indicating and Crew-Alerting System (EICAS) when climbing through flight level FL380. At the same time, the aircraft's slip/skid indication deflected to the full right position on the primary flight display (PFD). The PFD airspeed display then indicated that the aircraft was approaching the overspeed limit and the stall speed limit simultaneously. The aircraft pitched up and climbed to approximately FL410, and the indicated airspeed decreased from 270 kts to 158 kts. The stall warning and stick shaker devices also activated.

Because of corrective action by the crew, using the Standby Attitude and Air Data Reference Unit (SAARU) data, this incident did not materialize into an accident. The aircraft returned to Perth, where an uneventful landing was completed. The aircraft's flight data recorder (FDR), cockpit voice recorder, and the ADIRU were removed for examination. The FDR data indicated that, at the time of the occurrence, unusual acceleration values were recorded in all three planes of movement. The acceleration values were provided by the aircraft's ADIRU to the aircraft's primary flight computer, autopilot, and other aircraft systems during manual and automatic flight.

The B777 ADIRU has six sensing axes, skewed 57 degrees apart (along the face normals of a semi-dodecahedron). These are algebraically combined and resolved into the three principle axes. The ADIRU is fail-op:fail-op:fail-safe, but only sequentially (i.e., the first fault has to be accommodated before the next fault occurs). The automatic accommodation is to identify the faulty axis and lock it out of the computations until repaired. In this case, after 4 years with the first fault not being repaired, a second fault occurred, which then uncovered a software-design fault that had been masked until a post-cert change exposed it. This erroneously unlocked the first fault's lockout. The four years of flying with a failure was well beyond the requirement placed on the design.

As shown in figure 2, subsequent examination of the ADIRU revealed that one of several accelerometers had failed at the time of the occurrence and that another accelerometer had previously failed in June 2001 but was not repaired. When the accelerometer failed in 2001, it was excluded by the ADIRU health checks. This fault was not repaired because, by the multiply redundant design, it was not thought to be critical and did not generate a maintenance message on the EICAS, although the fault was recorded in the ADIRU. This incident occurred when a second accelerometer failed in 2005. An ADIRU software error introduced during a software update allowed bad data from the first (2001) failed accelerometer to be used instead of continuing to exclude it. The bad data then caused the flight-control system to command an abrupt nose-up pitch. Testing of the software update had considered only a stuck-at-zero fault and not the hard-over fault that occurred.

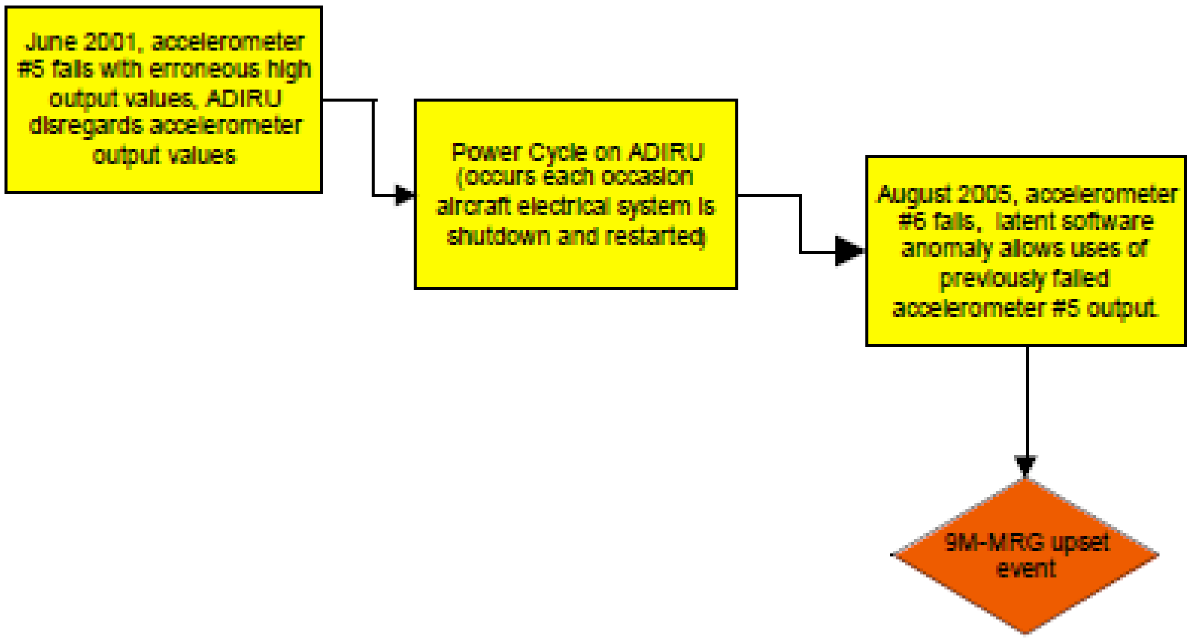

June 2001, accelerometer #5 fails with erroneous high output values, ADIRU disregards accelerometer output values
Power Cycle on ADIRU (occurs each occasion aircraft electrical system is shutdown and restarted)
August 2005, accelerometer #6 fails, latent software anomaly allows uses of previously failed accelerometer #5 output.
9M-MRG upset event

Figure 2. 9M-MRG upset sequence of events

Prior to this incident, the FAA had issued AD 2005-10-03 (on 06/15/2005) requiring that ADIRU software be modified from p/n 3470-HNC-03 to -06 or -07 versions. Because of this incident, the FAA then issued emergency AD 2005-18-51, effectively undoing AD 2005-10-03. Operators were required to backtrack to the -03 version. Because this backtracking then reintroduced the original fault of potentially using erroneous data, the FAA also required that the minimum equipment list (MEL) be amended to require an operational SAARU and that Boeing update the operations manual. An -08 software version was rolled out to the entire fleet in early 2006 as a mandatory change, and the SAARU MEL requirement rolled back.

ATSB quote: "The certification of the ADIRU Operational Program Software was dependent on it being tested against the requirements specified in the initial design. The conditions involved in this event were not identified in the testing requirements, so were not tested."

<u>3.1.3 Turkish Airlines B737, Amsterdam 2009</u>

This aircraft stalled on approach and crashed because of erroneous left-side radio altimeter data [6]. This caused early retard flare mode to be engaged; the aircraft pitched up and stalled, then crashed on landing. There were nine fatalities. The investigation made 11 recommendations.

During approach at 1950 feet, a faulty captain-side (left) radio altimeter suddenly flipped to reading -8 ft, causing the auto throttle to (correctly) decrease engine power to the "retard flare" low power setting. This should be used only when on flare-out at 27 ft, just prior to touch down. This failure should have logged an error and transferred data sourcing to the right-side radio altimeter, which was reading correctly, but it did not and continued to be the source of altitude data for the auto throttle and other systems. The crew would have had no understanding of the conflicting effect on the auto throttle of an undetected failure to one radio altimeter while the other continued to perform correctly.

The pilot flying was the first officer (right seat) and, therefore, the autopilot in control was also on the right-side. The right-side autopilot received altitude data from the still-functioning right-side radio altimeter and, therefore, attempted to keep the aircraft flying on the glide path for as long as possible. This meant that, unnoticed by the crew, the aircraft's nose continued to rise, creating an increasing AOA of the wings while the autopilot attempted to maintain lift as the airspeed reduced.

The aircraft continued to descend and slow down until the stick shaker activated; the captain then took control. There appeared to be some confusion between the captain and the first officer in this transfer of control. The first officer released the throttles, which caused the auto throttle to return the throttle correctly setting to idle according to its view of the world. There was some delay until the captain disconnected the auto throttle and commanded full thrust, but by this time, there was insufficient altitude (350 ft) or forward speed available to affect a recovery.

The basic cause was that the left-side radio altimeter failed undetected, therefore defeating the dual redundancy. The deeper cause was the system design that allowed the auto throttle and autopilot to operate from disagreeing altitude sensors. Everything else functioned as designed.

Had the crew been aware of the failure mode and its implications for this system design, recovery action could have been taken in time to avoid the accident

3.1.4 Leisure Airways A320, Ibiza 1998

This is an example of a Byzantine failure in which two receivers read the same signal but interpreted the signal differently [7]. The A320 brake system is a dual-redundant system comprised of two brake and steering control units (BSCU), each comprised of a command/monitor (COM/MON) configuration and associated hydraulic valves. Brake force is derived from brake calipers driven from the aircraft hydraulic system. In this case, the COM/MON disagreement was simultaneous in both BSCU channels' reading of the autobrake select switch. This switch is a spring-loaded-to-off pushbutton located in the cockpit and operated by the crew prior to landing. This switch selects the rate of automatic braking to be applied after touchdown. Three values are available to the crew, LO/MED/HI, selected according to the reported runway surface conditions.

Because the switch is spring loaded, the duration of the signal provided depends on the crew action. In this case, the duration was sufficiently long for one side to register it immediately and the other to register one minor cycle later because the clocking cycles of all four channels are asynchronous. Such an effect is typical in which two asynchronous systems read a common signal; they will occasionally read different values. The comparison functions built into the BSCU detected the disagreement and put both BSCUs offline. This left only manual pedal braking available to the crew. In this instance, the spring-loaded Norm/Alt hydraulic pressure source selector valve was also jammed in Normal by a frozen water-detergent mix in its lower cover. Because the BSCUs were both failed, the Normal hydraulic pressure source channel was switched off, meaning that switchover to manual braking taken from the Alternate pressure source could not occur. The overall result was total loss of braking.

The aircraft overran the runway and suffered damage, but there were no fatalities. The investigation authority made nine recommendations: eight to Airbus and one to the French certification authority DGAC. However, none of these recommendations state that Byzantine fault tolerance is needed. In fact, the word Byzantine does not appear anywhere in the report.

3.1.5 Air France A340, North Atlantic 2009

The French investigation authority (BEA)[4] determined that the probable cause was unreliable airspeed indication from frozen pitot tubes. The speed calculated by the Air Data Module from the pitot tubes is distributed to many other systems.

The determination of unreliable airspeed caused the autopilot to disconnect, and the automatic flight control system switched to Alternate Law. In this mode, there is no automatic stall protection (α-prot). The crew was seemingly at a loss to understand the situation and did not follow the documented unreliable airspeed procedures. The aircraft entered a high-altitude stall from which it could not recover. This loss of SA by the crew caused loss of control (LOC) in an unrecognized stall condition. The investigation authority noted the lack of crew training and ability in manual flying. This example illustrates how a fault in one part of a system can reverberate through many others, ultimately causing an accident.

3.1.6 Virgin Atlantic A340 En-Route Hong Kong to London Heathrow 2005

Some 11 hours after takeoff, at approximately 0330, with the aircraft in Dutch airspace and at Flight Level 380, the No. 1 engine lost power and ran down [8]. Initially, the pilots had suspected a leak and had emptied the contents of the fuel tank feeding the No. 1 engine, but a few minutes later, the No. 4 engine started to lose power. At that point, all the fuel cross-feed valves were manually opened, and the No. 4 engine recovered to normal operation. The pilots then observed that the fuel tank feeding the No. 4 engines was also indicating empty, and they realized that they had a fuel-management problem. Fuel had not been transferring from the center, trim, and outer wing tanks to the inner wing tanks that feed the engines, so the pilots attempted to transfer fuel manually. Although transfer was partially achieved, the expected indications of fuel transfer in progress were not displayed; the commander decided to divert to Amsterdam Schiphol Airport, where the aircraft landed safely on three engines. The fuel transfer had in fact been taking place after the transfer valves were opened manually, but there was no indication of this. The crew had to assume that the fuel was not being transferred and that it may imminently run out.

The investigation determined that the following causal factors led to the starvation of inner fuel tanks 1 and 4, and to the subsequent rundown of engines 1 and 4;

- The automatic transfer of fuel within the aircraft stopped functioning because of failure of the discrete outputs of the master fuel control and monitoring computer (FCMC).
- Because of FCMC ARINC data bus failures, the flight warning system did not provide the flight crew with any timely warnings associated with the automated fuel control system malfunctions.
- The alternate low-fuel-level warning was not presented to the flight crew. The Flight Warning Computer disregarded the Fuel Data Concentrator data because its logic determined that at least one FCMC was still functioning.
- The health status of the slave FCMC may have been at a lower level than that of the master FCMC, therefore preventing the master FCMC from relinquishing control of the fuel system to the slave FCMC when its own discrete and ARINC outputs failed.

The FCMCs had been reset at separate times on the previous flight sector from Sydney to Hong Kong. During the preflight preparation period for the flight to London, there was one FCMC2 and one FCMC1 failure. The flight crew successfully reset both computers on each occasion. Each FCMC consists of three processors—two operating in a COM/MON configuration, with the third as an integrity checker. COM/MON disagreement hands over control to the second FCMC. This had been a recurring problem since initial certification (software version 4.1, July 2002), and several software versions had been certified to try and resolve it. The version on this aircraft was Flight Load 7 certified in November 2003. Flight Load 8, certified in February 2005, was intended to fix the continuing COM/MON disagreement failures but had not yet been deployed. During taxi-out, FCMC1 again failed, but attempts to restart it were unsuccessful, so the aircraft continued with just FCMC2 operational.

Each FCMC maintains an internal 8-level current health status, level 0 being fully healthy and 1–7 representing various internal failures. The two FCMCs communicate their health levels through an ARINC bus and a set of analogue discrete lines. It appears that the aircraft suffered two failures, FCMC1 then FCMC2. Subsequent testing of the two FCMCs on the ground and in a test flight failed to reveal any faults. They also reacted as expected to injected faults. The system design was deficient because the crew did not get any notification that fuel transfers had ceased until one engine ran out of fuel. The assumption seems to have been that the aircraft was dispatchable with one failed FCMC because a further failure could be overcome by manual control of the fuel valves.

This logic neglected to consider that to intervene, the crew must be given a warning of the failure, which did not happen in this case.

The two engines flamed out because the fuel tanks supplying those engines were empty, and those for the other two engines were very low. Two FCMCs pump fuel between tanks. FCMCs cross-compare, and the "healthiest" one drives the data bus. Both FCMCs had known faults but complied with the minimum capabilities required for flight. One of the faults in the one judged healthiest was the inability to drive the data bus. Therefore, although it gave correct commands to the fuel pumps (there was plenty of fuel distributed in other tanks), these were never received.

3.1.7 Air Asia A320, Karimata Strait Indonesia, 2014

Repeated failure of both rudder travel limiters (RTLs) caused multiple ECAM warnings. In attempts to clear the warnings, the crew cycled FAC power repeatedly, eventually causing the FCS to revert to Alternate Law. Crew confusion and incorrect control actions led to high bank angle, a stall, and unrecoverable LOC. The aircraft descended at 12000ft/min with +40deg AOA at 100–160kts IAS. The crew did not appear to use Integrated Standby Instrument System standby air data to manually recover straight and level attitude [9].

RTLs had previous repeated intermittent failures that had not been diagnosed or repaired. Maintenance had simply power cycled and returned to service. The initial RTL failure was traced to cracked solder joints in both RTLs.

This was potentially a recoverable situation but was compounded by poor crew resource management, poor crew training on upset management, and crew inability to take control and fly manually.

The aircraft crashed, and there were 162 fatalities (all on board). Five recommendations were made.

3.1.8 China Airlines A300, Nagoya Japan 1994

In the China Airlines A300 crash at Nagoya[10], an initial pilot error accidentally activated the go-around switch while on approach. He tried to force the aircraft down manually but forgot to disconnect the autopilot, which was commanding pitch up. When the crew eventually realized their mistake and disconnected the autopilot, the resulting rapid pitch-up caused a stall, and the aircraft crashed tail-first. In this case, nothing failed; the autopilot reacted as designed. However, from a systems viewpoint, a requirements error occurred. This is because the autopilot design did not account for the possibility of the crew overriding the pitch-up command and then suddenly releasing yoke pressure while being close to the ground.

3.1.9 U.S. Customs and Border Patrol Predator-B, Nogales AZ, 2006

A control-station malfunction caused the pilot to transfer aircraft control from operator position PPO-1 to a second position PPO-2, which is usually used by the camera operator. A mistake in aligning the control column positions of PPO-1 and PPO-2 prior to the transfer caused the fuel valve to be shut off, causing the engine to run out of fuel and the aircraft to crash [11]. The transition from PPO-1 to PPO-2 was made in flight because of a failure (lockup) of PPO-1, a notuncommon occurrence. At the time of the transfer, the control lever on PPO-2 was set to a “fuel valve cutoff” position. When transfer had been performed, the fuel valve closed, and engine power was lost. The incident was attributed to operator error in not matching the control lever positions on PPO-1 and PPO-2 just prior to transition, as was required by the checklist and standard operating procedure (SOP).

The loss of engine power also resulted in a loss of electrical power except for the standby battery. On battery power, the transponder was not operational, so no returns were given to interrogations. The loss of engine power caused the Predator to lose height, eventually dipping into mountainous terrain where it was lost to radar surveillance. The operator realized what he had done and decided to put the Predator into its lost link profile by turning off PPO-2. The aircraft did not go into the lost link profile as he had expected because it had no power and was already descending, although the operator did not know this; it was a case of mode confusion. The loss of transponder and radar returns meant that nobody knew where it was. The possibility that it was flying a lost-link profile meant one had to assume that it was a potential hazard to other aircraft.

The operators were blamed for the incident in the National Transportation Safety Board (NTSB) report (probable cause in NTSB parlance) for not following SOP, as was the case. This example again illustrates a defective system design and missing requirement that allowed an unsafe condition to arise by allowing handover with a misplaced control lever setting and a susceptibility to a single-point failure.

3.1.10 XL Airways A320 Delivery Flight, Canet-Plage 2008

The accident occurred during a non-revenue check flight of an A320 aircraft prior to return to its owner, Air New Zealand, from its lessee, XL Airways [12]. This accident was attributed to prior aircraft cleaning procedures following a repaint to Air New Zealand colors, which had forced water into the AOA sensors. The cleaning process did not protect the sensors from water ingress. Water that had entered the #1 and #2 sensors during the cleaning operation froze in flight at the normal cruise angle. The AOA sensor is a mechanical vane on the outer skin of the aircraft with internal heating on the vane only. Unknown to the crew, they remained stuck at the same cruise angle for the rest of the flight.

The A320 has fly-by-wire flight controls. The aircraft is flown using two sidesticks whose movements are transmitted in the form of electrical signals to computers that transform them into orders to the actuators of the various surfaces. The laws that govern these transformations are called flight control laws. On the A320, in nominal operation, the flight control law is called normal law. Under certain conditions, it can be replaced by two degraded control laws: alternate law and direct law.

Normal law offers protections in attitude (the pitch and bank values are limited), in load factor, in overspeed, and at high AOA. Pitch trimming is ensured automatically by the auto-trim. Turn is coordinated, with the rudder minimizing the sideslip through the yaw damper function. The sidesticks control the load factor according to the normal aircraft axis and the roll rate.

In alternate law, the sidesticks control the load factor according to the normal aircraft axis as for normal law, but with fewer protections. In roll, they directly control (as they do in direct law) the ailerons and the spoilers. The passage from normal law to alternate law is accompanied by a MASTER CAUTION warning, SINGLE CHIME aural warning, and an ECAM message. The green dot indicators for attitude protection are replaced by amber crosses. In alternate law, when the landing gear is extended, the pitch-control law passes to direct law.

In direct law, there is no automatic pitch trimming. Trimming is performed manually using the trim wheel. The orders to the flight-control surfaces are a direct function of the control inputs on the side-stick. The passage to direct law is accompanied by a MASTER CAUTION warning, SINGLE CHIME aural warning, and an ECAM message. The display on the PFD is identical to that in alternate law with, in addition, the USE MAN PITCH TRIM message in amber. When the aircraft exceeds certain attitude thresholds, roll, AOA, or speed, the system uses a specific law. The display on the PFD is identical to that for alternate law.

A "low speed in landing configuration" check was begun on approach at 4000 ft with the intention of performing a subsequent missed approach and go-around, and then departing for Frankfurt. The FCS passed into direct law, and the aircraft stall warning then sounded because the #3 AOA sensor was still working. The crew attempted recovery in the standard manner of increased thrust and pitch down. The horizontal stabilizer was at full-up position, so the pitch-down input was insufficient to decrease AOA, and a stall occurred. They were not able to correct the stall, lost control, and the aircraft crashed.

3.1.11 British Airways B747-400/RR Incident, South Africa 2009

On May 11, 2009, a Boeing 747-400 aircraft (G-BYGA) equipped with Rolls Royce engines operated by British Airways was involved in a serious incident during takeoff from O.R. Tambo Airport at Johannesburg, South Africa [13].

During the takeoff roll, the No. 3 Engine thrust reverser (TR) EICAS amber message displayed on the P2-Pilots Center Instruments Panel before V1 at approximately 125.6 kt and shortly thereafter. The No. 2 Engine Thrust Reverse EICAS amber message displayed at approximately 159.9 kt prior to a rotation speed (VR) of 168 kt. At this stage, the Group A leading edge (LE) flaps retracted automatically as designed with the aircraft still in the ground mode. The aircraft rotated at approximately 173 kt with 20 units of flaps selected and became airborne at approximately 176 kt. The stick shaker subsequently activated at 176 kt intermittently for 8 seconds within a period of 15 seconds, and significant aircraft buffeting was experienced. To counteract the stall warning and buffeting, the pilot flying (who also had aerobatic flying experience and being familiar with aircraft buffeting) continued to fly the aircraft with the Pilot in Command calling out the aircraft heights AGL. The undercarriage was selected up at a computed airspeed (CAS) of 177 kt, and the Group A LE flaps immediately extended automatically. The stick shaker stopped at a CAS of approximately 186 kt. The Group A LE flaps had been in the retracted position for approximately 23 seconds during the occurrence. After the auto re-extension of the Group A LE flaps, the aircraft's performance returned to normal.

The subsequent investigation by South African authorities found that the flap control unit (FCU) received a signal (which it supposed was valid) from the TR controller indicating that #2/3 engines TR were in transit. The TRs did not actually transit according to the FDR evidence. The cause of this signal was never identified, but it was later found during an on-ground inspection that the

#2/3 TRs were not fully locked in place when retracted. The flap controller responded to the signal as designed and retracted the LE flaps. This was a hazardous action in the takeoff flight phase. This resulted in losing lift during takeoff rotation and numerous stall warnings. The temporary fix by Boeing was to disable the auto-retract on TR unlock function of the FCU on B747-400/RR. The FCU designers did not anticipate the possibility that an upstream subsystem could send a signal that would cause their subsystem to be hazardous when performing as expected and per requirements. This seems to be a recurring theme with other previous incidents. The integrator did not consider the lack of vehicle-level safety analysis, or it is so improbable that the possibility can be ignored (it was not that small after all). The deepest root cause therefore appears to be that the requirement to retract flaps on the TR in-transit signal is incomplete and should have had a qualifier for flight phase.

3.1.12 Tarom Flight 381

According to an accident report [14], an aircraft on approach to Paris Orly airport suddenly started to climb, adopted a steep pitch attitude, and stalled. The crew managed to recover control of the aircraft and came around to land. There were no injuries, and there was no damage to the aircraft. However, this incident made a strong impression on the passengers. The aircraft was an Airbus A310, registration YR-LCA, being operated as flight 381 by Romanian Tamron airlines. The weather conditions were excellent.

The proximate cause was “automation surprise.” Subsequently, this incident was used as a case study in examining the complementary use of model checking and human-performance simulation [15]. This study found the automation surprise that created the Tarom flight 381 incident and identified additional automation surprises associated with that flight logic.

The important events in this incident were:

- While on automatic approach, ground control asked the aircraft to shorten its path, which led to an ILS interception closer to the runway than provided for by standard procedure.
- Per aircraft system logic, the glide, encountered before the localizer, was not automatically captured.
- When flaps were selected at 20 degrees, the speed was slightly greater than V MAX , which activated speed protection, leading to reversion of VS mode to LVL CHG mode.
- Because the selected altitude was greater than that of the aircraft, the auto-throttle commanded an increase in thrust. The pilot maintained the aircraft on descent. The automation and the pilot were now at cross purposes.
- The pilot continued to counter the automation by continuous effort on pitch and by temporarily holding the thrust levers in the idle position. He neither corrected trim, which remained on pull-up stop, nor disconnected the auto-throttle.
- Under the effect of strong drift on full and rapid rolls, the AOA sensors were disturbed, which led to automatic disconnection of the two pitch-trims. The auto-throttle was inhibited for the same reasons.
- Because of the dynamic of the aircraft’s movements, the stall warning and the stick shaker did not function appropriately.
- The flight crew regained control of the aircraft after the stall.

At several points in the CVR, phrases like “qu'est-ce qu'il fait” and “qu'est-ce qu'il a” (both of which can be translated as “what is it doing,” and the latter can be more closely translated to “what is it doing wrong”) indicate that the crew did not understand what the automatics were doing.

3.2 INTERNAL INVESTIGATIONS

Honeywell has a root cause corrective action (RCCA) process to investigate safety incidents that occur in operations. One example is described in section 3.2.1.

3.2.1 Honeywell Data—Cockpit Displays Blanking

Multiple display blanking events on business jet pilot display units (PDUs) and Multifunction Display Units occurred during 2014–2015. These occurred on several flights, aircraft types, and software loads. The blanking events were temporary, approximately 1–2 seconds’ duration, non-cyclic, one-side only, and displayed no hazardously misleading information (HMI). Only a portion of each display was lost, whereas other windows on the display remained active. This was considered a minor safety issue because of the absence of HMI.

The investigation took the recorded Integrated Modular Avionics (IMA) health-monitoring data and played it back on a simulator using instrumented PFD code. The determination was that the consumption of erroneous or inconsistent Level-C flight management system (FMS) map data caused a divide by zero floating point exception and display reset. This occurs when:

- The aircraft is precisely over a waypoint.
- The difference between the FMS reported position and another present position data source is close to zero (<100mm)
- The crew has selected the Horizontal Situation Indicator Arc and Flight Plan display modes

Code reviews are routinely conducted during software development in which exception handling for “illegal arithmetic operations” is sought. It appears that in this case, the absence of defensive code was not flagged. The RCCA taken was to implement a software fix to trap a zero denominator. In addition, three process-improvement initiatives with associated design procedures were launched.

- SW-HW coupling/partitioning addressed early in the design phase with targeted requirements generated
- SW-SW control-data coupling addressed early in the design phase with targeted requirements generated
- Improved module-integration tooling and processes to prevent errors

4. SURVEY OF TOOLS AND TECHNIQUES

To establish the start-of-the-art technologies relating to system and safety engineering, the Honeywell team performed a survey of current engineering tools and related technologies. The following provides short summaries and links to commercially licensed and open-source model-based system engineering (MBSE), model-based safety analysis (MBSA), formal methods, and related tools and technologies. Free and open-source tools are indicated by (OS), although the license particulars vary. All others require some type of paid license. This tool survey and other information gathering were used as background material for forming the ideas presented in section 10.

4.1 MBSA TOOLS

This section lists tools that are specifically targeted at MBSA, although the claim is exaggerated in some cases, given what safety analysis really is. What is termed MBSA is probabilistic risk analysis (PRA) rather than actual safety analysis, similar to ARP-4761.

4.1.1 Architectural Analysis and Design Language EMV2 (OS)

[http://www.aadl.info/aadl/currentsite/]

Developed by a Society of Aerospace Engineers (SAE) International-sponsored committee of experts, Architectural Analysis and Design Language (AADL) [16, 17] was approved and published as SAE Standard AS-5506 in November 2004. Version 2.1 of the standard was published in September 2012. The AADL is designed for the specification, analysis, automated integration, and code generation of real-time performance-critical (timing, safety, schedulability, fault tolerance, security) distributed computer systems. It provides a new vehicle to allow analysis of system designs (and system of systems) and integrated safety analysis prior to hardware and software development. It supports a model-based, model-driven development approach throughout the system life cycle. The language is supported by the OSATE2 editor (https://wiki.sei.cmu.edu/aadl/index.php/Osate_2) based on the Eclipse open-source Integrated Development Environment (IDE) platform. This tool supports the specification of cyber-physical systems in AADL and fault tree analysis (FTA), and Failure Mode And Effect Analysis (FMEA), using the AADL Error Model Annex (EMV2) annotations [18].

The tool allows the user to define the error characteristics and failure modes of the components comprising the system, and the flow of errors through the system, using the connectivity defined for the system. These are used to construct the final fault tree from a Boolean and/or expression of the basic events. The FTA creates files in OpenFTA format to display a graphical fault tree. The analyst is responsible for defining the composite error behavior Boolean equations of the fault tree. The basic event probabilities are specified in the AADL of the components.

A recent update (see the instructions at https://github.com/juli1/emfta) provides a means to produce the graphical style of output in image-file form within the tool and a tabular format using the EMFTA export facility. Both formats show the top-level event probability. The exported EMFTA file must be imported to the Eclipse Mars modeling tool (https://www.eclipse.org/) with the EMFTA plug-in installed to generate the graphical fault tree representation. Both OSATE2 and Eclipse Mars tools require the EMFTA add-on to be installed. An example is provided for a Wheel Brake System similar to that shown in ARP-4761 at wiki.sei.cmu.edu/aadl/index.php/ARP4761_-_Wheel_Brake_System_%28WBS%29_Example.

AADL and the Error Annex EMV2 are only able to create fault trees from composite error behavior equations and, therefore, cannot claim to do safety analysis. Composite error equations express the fault tree for a top-level event (TLE) as a Boolean expression. The fault tree can therefore be considered a 1:1 mapping of the expression with no new information added. Because

AADL/EMV2 has no notion of system function and neither Assume/Guarantee Reasoning Environment (AGREE) nor Behavioral Language for Embedded System Software (BLESS) are integrated with it, the construction of the Boolean equation depends entirely on the knowledge of domain experts, just as in conventional FTA.

4.1.2 Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS)

[http://www.hip-hops.eu/]

The HiP-HOPS tool was developed at the University of Hull. It supports state-of-the-art model-based techniques and application of search, meta-heuristics, and sophisticated model-based analysis algorithms on both sides of the "V" engineering lifecycle of a complex system. This includes both the refinement of dependable designs from requirements to detailed SW-HW architecture and the bottom-up verification of dependability. This work has achieved substantial international recognition; more than 100 papers have been published or presented on topics related to this work. In the context of a string of recent European projects (ATESST, ATESST2, and MAENAD), HiP-HOPS has contributed to the specification of the error-modeling capabilities of EAST-ADL, an emerging architecture-description language developed as an industry standard for the design of vehicle-control systems. HiP-HOPS today is widely recognized as a state-of-the-art technique in the area of dependability analysis.

Hip-HOPS depends on block diagrams and the connectivity of blocks created in Simulink, to which the error annotations (failures types, probability, flow through) are added by a custom Matlab Graphical User Interface (GUI) provided by the tool. The connectivity of the Simulink diagram defines the error propagation paths between components. Components are defined as functionally empty blocks, only including the ports, the errors sources, and propagations. These are the basic events of the final fault tree.

Each top-level hazard is defined in the model by a composite error behavior Boolean expression of basic events representing the Boolean logic of the fault tree for the hazard. The analyst is responsible for providing these Boolean equations. The resulting fault tree output is this equation with the basic event probabilities rolled up. There is no assistance provided by the tool to develop these expressions, check for errors, or check for completeness because the tool has no knowledge of the intended system function. The analyst must define them from an intimate knowledge of the intended system function.

Hip-HOPS is only able to create fault trees from composite error behavior equations; therefore, as for AADL/EMV2, Hip-HOPS cannot claim to do safety analysis. Composite error equations express the completed fault tree in Boolean form and, therefore, can be considered as a 1:1 mapping with no new information added.

4.1.3 COMPASS (OS with geographic restrictions)

[http://compass.informatik.rwth-aachen.de/]

COMPASS: Correctness, Modeling and Performance of Aerospace Systems. This European project integrates behavioral and safety analysis, targeting a simpler variant of AADL. The benefit of COMPASS is that it goes beyond AADL's disjoint modeling of behavioral and fault domains.

The project was sponsored by the European Space Agency (ESA). COMPASS allows the definition of the functional behavior of a system and then an addition of error behavior through fault injections. The SLIM syntax used by COMPASS is similar to AADL, with some limitations. The COMPASS tool provides a rudimentary GUI [19] but no integrated editor. This means using a generic text editor without syntax highlighting supported. Distribution is free but restricted to citizens of the ESA member states. The tool distribution is for Linux only. It was run on Debian and Ubuntu varieties of Linux. The standard distribution consists of a VMWare (www.vmware.com) Ubuntu Linux Virtual Machine (VM) image that can be run as a VM under Windows. The image contains the tool, documentation, some examples, and a pre-configured version of Ubuntu. Syntax and use are described in some papers [19–21] and in the documentation [22–24].

COMPASS SLIM models require that every component communicate over a bus; therefore, every model must contain a bus component declaration. This is purely a syntactical requirement of the SLIM language, unrelated to the physical architecture. The necessity for a bus component is implied by being used in the example models but is not specifically noted as a requirement in the user manual.

4.1.4 FAME (OS with geographic restrictions)

[https://es-static.fbk.eu/projects/fame/]

FAME: Failure and Anomaly Management Engineering. This is a follow-on to the COMPASS program focusing on safety. The FAME Project [25] is an international research project for developing a fault detection, isolation, and recovery (FDIR) development and V&V process. It is based on the previous COMPASS project, which it extends by new functionalities. The project was started in July 2012 and was concluded in May 2014. The consortium consists of the ESA (funder), Thales Alenia Space Italia (prime contractor), Thales Alenia Space France (industrial subcontractor), and Fondazione Bruno Kessler (FBK, Italy) (research subcontractor). The FAME toolset is distributed as a VMWare preconfigured image for Windows and as Debian/Ubuntu native Linux. FAME is subject to the same licensing conditions as COMPASS.

FAME adds timed failure propagations graphs (TFPG) to the COMPASS toolset. A TPFG is a directed graph (digraph) model that represents temporal progression of failure effects in physical systems [26] (i.e., a TPFG is a causal model that describes the system behavior in the presence of faults). These were originally developed for system fault diagnosis.

4.1.5 KB3 (OS)

[http://researchers.edf.com/software/kb3-44337.html]

The KB3 workbench [27], which EDF has been developing for approximately 15 years, allows automating dependability studies by capitalizing on the knowledge acquired on systems through knowledge bases written in an object-oriented modeling language (FIGARO). KB3 utilizes the Visual Figaro and YAMS tools, also available at the website.

4.1.6 AltaRica 3.0 (OS)

[http://altarica.labri.fr/wp/]

The objective of the AltaRica 3.0 project [28] is to design a new version of AltaRica (a modeling language designed by Laboratoire Bordelais de Recherche en Informatique in Bordeaux and some industrial companies approximately two decades ago) and to develop a complete set of authoring, simulation, and assessment tools to perform safety analyses. A supporting platform is available from http://openaltarica.fr/.

4.1.7 EDICT

[http://www.wwtechnology.com/default.htm]

With the EDICT tool, system engineers and architects are able to capture the important properties of a system as models of system architecture and behavior, then assess the safety and efficiency of architectural mechanisms employed. The tool plays a role in the overall V&V process for ensuring that requirements for dependability, safety, and error-handling requirements are achieved with traceable artifacts.

EDICT is able to import a system defined in AADL, but is not yet able to import AADL Error Annex annotations (EMV2). These must be added manually in the tool.

4.1.8 NuXMV (OS)

[https://nuxmv.fbk.eu/]

NuXMV [29] is a noncommercial or academic-use-only safety-analysis tool based on the earlier NuSMV tool. It has been superseded by xSAP. NuSMV is incorporated in the COMPASS and FAME tools.

4.1.9 xSAP (OS)

[https://es-static.fbk.eu/tools/xsap/]

xSAP [30] is a tool for safety assessment of synchronous finite-state and infinite-state systems. xSAP is licensed for non-commercial use by the FBK. It is based on symbolic model checking techniques. xSAP provides the following main capabilities:

- Library-based specification of faults, fault effects, and fault dynamics
- Automatic model-extension with fault specifications
- FTA and generation of minimal cut sets for dynamic systems, for both the monotonic and non-monotonic case
- FMEA
- Fault-propagation analysis based on TFPG
- Common cause analysis

4.1.10 Othello Contracts Refinement Analysis (OS)

[https://es.fbk.eu/technologies/ocra-othello-contracts-refinement-analysis]

Othello Contracts Refinement Analysis (OCRA) is a tool for the verification of logic-based contracts refinement for embedded systems. It supports the specification of components enriched with Othello contracts and the compositional verification of contracts refinement and implementation. It is built on top of the NuXMV model checker. OCRA also integrates with xSAP to develop fault trees based on either the contracts alone or extended by the use of fault-injection models.

In OCRA, contracts are composed of assumptions and guarantees represented in linear temporal logic (LTL) formulas expressed in an OCRA language. An assumption is the set of formal properties that an analysis block (e.g., an architectural component) expects to be provided by its environment, and a guarantee is the set of formal properties that the block will provide subject to the assumptions being satisfied. This paradigm allows contracts to be established at every hierarchy level of a proposed architecture and allows checking that all lower-level contracts are satisfied. This is accomplished through the formal discharge of proof obligations by the model checker (NuXMV).

The significant feature of OCRA that distinguishes it from other tools is the integration of functional and fault modeling into one environment. This is an important feature because safety is not only a function of reliability but also of the functional design.

A plug-in for AF3 [https://es-static.fbk.eu/tools/autofocra/] allows for the visualization of architectures initially constructed using the text format of an Othello System Specification (OSS).

An OCRA example is shown in section 6 with a design similar to the SAE ARP-4761 and SAE AIR-6110 Wheel Brake System examples [31]. This case study shows a method of capturing a system design in unified modeling language (UML)/system modeling language (SysML) (using Papyrus), automatically translating it into an OSS representation for OCRA, and checking that a selection of system-level hazards meets certification requirements.

In the present state of development, the OCRA tool is best suited to architectures that are purely logical (i.e., the case for most types of electronic hardware and all software). It is less suited to physical systems, such as a hydraulic or electrical, forming a closed-loop circuit because the return flow path aspect of such systems cannot yet be modeled.

The OCRA tool is also unable to represent nonlinear functions. This may be as simple as the product of two continuous variables, a type of function that is commonplace. Some work to overcome this constraint is in earlier stages but is not available. Currently, the work-around is to discretize one of the variables, but this is an approximation.

4.1.11 SafetyHAT (OS)

[http://www.volpe.dot.gov/infrastructure-systems-and-technology/advanced-vehicle-technology/safetyhat-transportation-system]

The Volpe transportation systems Safety Hazard Analysis Tool (SafetyHAT) is a software tool that facilitates hazard analysis using the STPA. STPA (from MIT) is a hazard-identification method based on a top-down system engineering approach and systems theory. Whereas some familiarity with STPA is expected before using this tool, one of the primary goals of SafetyHAT is to help safety analysts become proficient with the STPA method. SafetyHAT includes transportation-oriented guide phrases and causal factors that tailor the STPA method to transportation systems. SafetyHAT is based on an MS Access database and therefore relies on that being installed.

4.1.12 XSTAMPP (OS)

[http://www.iste.uni-stuttgart.de/se/werkzeuge/xstampp.html]

A second tool supporting System Theoretic Accident Model and Process (STAMP)/STPA is XSTAMPP (An eXtensible STAMP Platform)[32]. XSTAMPP is an open-source platform for safety engineering designed specifically to serve the widespread adoption and use of STAMP-based tools (STPA and causal analysis based on system theory [CAST]) in different areas. XSTAMPP includes three plug-ins: A-STPA (automated tool support for STPA), A-CAST (automated tool support for A-CAST), and XSTPA (extended approach to STPA). Moreover, XSTAMPP provides support to automatically transform the context tables into LTL formal specifications. XSTAMPP is written in Java based on the Eclipse Plug-in-Development Environment and rich client platform (RCP). The RCP architecture provides easy expandability and flexibility. XSTAMPP provides core functionality and components that make it easy to extend with new plug-ins. XSTAMPP was developed with the background to create a base platform support for safety engineering, which can be easily extended with new functions and approaches based on the STAMP model. The current version of XSTAMPP 2.0 is available as an open-source platform at http://sourceforge.net/projects/stampp/files/

4.1.13 QuantUM (OS)

[http://quantum-tool.com/]

QuantUM is a tool for model-based functional safety analysis from the University of Konstanz. QuantUM offers automated, model-based functional safety analysis (FTA, FMEA). QuantUM is not actively under development.

4.1.14 Medini™ Analyze

[http://www.ikv.de/index.php/en/products/functional-safety]

Medini Analyze is targeted at ISO 26262 automotive system developments and integrated in a single tool. The application of ISO 26262, published in November 2011, is mandatory for the analysis and development activities in the automotive domain. Medini Analyze is an integrated tool that efficiently implements core activities of the functional safety analysis and integrates them with the existing processes. Target users are safety managers, experts, development engineers, and quality managers involved in the development of electronic and software-based components in the automotive industry. Medini Analyze offers:

- An integrated solution for many of the ISO 26262 activities.
- Standard compliant work with minimal effort.
- Support from concept phase to production and operation.
- Numerous interfaces to other tools of engineering environment.
- Automatic assurance of the consistency of work products.
- High degree of reusability because of library, catalog, and template mechanisms.
- Automatic generation of the work products required by ISO 26262.
- Sophisticated support of assessments and reviews.

4.1.15 All4TEC

[http://www.all4tec.net/]

ALL4TEC is a software publishing company in functional validation, safety analysis, and system engineering of complex and interconnected systems.

4.1.16 Causalis (OS)

[http://www.causalis.com/]

Causalis is the technology transfer company of Prof. Peter Ladkin, University of Bielefeld, Germany. Causalis has developed two methods in-house: why-because analysis for the causal analysis of significant failures and accidents, and ontological hazard analysis for safety analysis during the forward development of critical systems from requirements to software.

4.1.17 Dymonda

The Dymonda tool implements the Dynamic Flowgraph Methodology (DFM). Dymonda provides the following facilities:

- Graphical DFM model construction
- Static and time-dependent probabilities
- DFM model analysis
- Inductive analysis
- Deductive analysis
- Quantification
- Exact results
- Prime implicant upper bounds
- Precision-specified approximations

DFM is an integrated methodological approach to modeling and analyzing the behavior of software-driven embedded systems for the purpose of dependability assessment and verification. The methodology has two fundamental goals: 1) to identify how events can occur in a system, and 2) to identify an appropriate testing strategy based on an analysis of system functional behavior. To achieve these goals, the methodology employs a modeling framework in which models expressing the logic of the system being analyzed are developed in terms of causal relationships between physical variables and temporal characteristics of the execution of software modules.

These models are then analyzed to determine how a certain state (desirable or undesirable) can be reached. This is accomplished by developing timed fault trees, which take the form of logical combinations of static trees relating the system parameters at different points in time. The resulting information concerning the hardware and software states that can lead to certain events of interest can then be used to increase confidence in the system, eliminate unsafe execution paths, and identify testing criteria for safety-critical software functions [33].

4.1.18 BLESS

[http://bless.santoslab.org/node/1]

BLESS is a behavioral Annex to AADL and a supporting proof environment, [34, 35]. BLESS performs virtual integration of the components of an architecture by adding behavioral specifications to an AADL architectural model. BLESS specifications are formally validated to show that the functional properties of an architecture are satisfied compositionally by functional properties of the contained components. The use of this language is described in more detail in a medical device context by Larson et. al. [36].

4.1.19 AGREE [http://loonwerks.com/tools/agree.html]

AGREE is a compositional, assume-guarantee-style model checker for AADL models. It is compositional in that it attempts to prove properties about one layer of the architecture using properties allocated to subcomponents. The composition is performed in terms of assumptions and guarantees that are provided for each component. Assumptions describe the expectations the component has on the environment, and guarantees describe bounds on the behavior of the component. AGREE uses k-induction as the underlying algorithm for model checking.

4.2 SAFETY CASE TOOLS

This section lists tools targeted to the construction and organization of a safety or assurance case. These are used primarily as a framework to organize and cross reference the arguments and evidence comprising the safety case.

4.2.1 ASCE

[http://www.adelard.com/asce/choosing-asce/index.html]

Adelard's ASCE is a commercial system for the development and management of assurance cases and safety cases. It is in use in many industry sectors worldwide.

4.2.2 Certware (OS)

[http://nasa.github.io/CertWare/index.html#introduction]

Kestrel Technology, LLC, is developing a prototype extensible workbench to develop, maintain, and analyze safety cases—a specialized form of dependability cases. The CertWare workbench contributes several core modules supporting safety case models, and extends these with service-based APIs for plugging new capabilities into the workbench to process these models.

4.2.3 Goal Structuring Notation (OS)

[http://www.goalstructuringnotation.info/about]

Goal Structuring Notation (GSN) is not a tool as such; it is a graphical method for reasoning about safety or assurance cases, as they are now becoming known. GSN is for the capture of safety case arguments and evidence. GSN is overseen by the Goal Structuring Notation Working Group, organized and maintained by the University of York. The Goal Structuring Notation Working Group consists of experienced practitioners who have used and continue to use GSN on major projects. The main objective of this site is to disseminate information and resources about GSN and to serve as a point of reference for resources and the GSN Standard.

4.3 FORMAL METHODS TOOLS

Several formal methods tools that may be used in support of MBSE are listed, although they do not specifically target MBSA.

4.3.1 CVC4 (OS)

[http://cvc4.cs.nyu.edu/web/]

CVC4 is an efficient open-source automatic theorem prover for Satisfiability Modulo Theory (SMT) problems. It can be used to prove the validity (or dually, the satisfiability) of first-order formulas in a large number of built-in logical theories and their combination.

4.3.2 YICES (OS)

[http://yices.csl.sri.com/]

Yices 2 is an efficient SMT solver that decides the satisfiability of formulas containing uninterpreted function symbols with equality, linear real and integer arithmetic, bit-vectors, scalar types, and tuples.

4.3.3 Z3 (OS)

[https://github.com/Z3Prover/z3]

Z3 is a high-performance theorem prover developed at Microsoft Research.

4.3.4 Event-B/Rodin (OS)

[http://www.event-b.org/]

Event-B (and Rodin, its supporting Eclipse-based IDE) is a formal method for system-level modeling and analysis. Key features of Event-B are the use of set theory as a modeling notation, the use of refinement to represent systems at different abstraction levels, and the use of mathematical proof to verify consistency between refinement levels. The Rodin platform is an Eclipse-based IDE for Event-B that provides support for refinement and mathematical proof. The platform is open source, contributes to the Eclipse framework, and is further extendable with plugins.

4.3.5 UPPALL (OS)

[http://www.uppaal.com/index.php]

UPPAAL is an integrated tool environment for modeling, simulation, and verification of real-time embedded systems. Typical application areas of UPPAAL include real-time controllers and communication protocols in particular—those for which timing aspects are critical.

4.3.6 PVS (OS)

[http://pvs.csl.sri.com/]

PVS is a verification system (i.e., a specification language integrated with support tools and a theorem prover). It is intended to capture state-of-the-art in mechanized formal methods and to be sufficiently rugged for use in significant applications. PVS is a research prototype.

4.3.7 SCADE

[http://www.esterel-technologies.com/products/scade-suite/]

SCADE Suite® is a model-based development environment dedicated to critical embedded software.

4.4 MISCELLANEOUS MBSE TOOLS ,PLATFORMS, AND LANGUAGES

4.4.1 OpenFTA (OS)

[http://www.openfta.com/default.aspx]

OpenFTA is an open-source tool for fault tree generation as an alternative to commercial tools, such as CAFTA and Relex. It allows the user to construct, modify, or analyze fault trees or import the required information from other tools. The OSATE FTA tool outputs a fault tree file in OpenFTA format for viewing the fault tree in the more traditional graphical format. The tool has not been maintained for some years.

4.4.2 PolarSys (OS)

[https://www.polarsys.org/]

PolarSys is a platform rather than a tool. It supports an Eclipse Industry Working Group created by large industry players and by tools providers to collaborate on the creation and support of Open Source tools for the development of embedded systems. For example, Papyrus is the PolarSys solution for SysML and UML modeling. It relies on the underlying Eclipse platform and on other Polarsys and Eclipse-based offerings to fulfill lifecycle integration needs, such as a C/C++ IDE, source control (e.g., Git), and reporting. Through its integrated support for UML and SysML modeling, Papyrus provides the basis for the adoption and use of Model-Driven Engineering and MBSE. Papyrus supports the following modeling standards versions: UML 2.5, OCL 2.3.1, fUML1.1, ALF 1.0.1, MARTE 1.1 (incubation), SysML 1.2, EAST-ADL (incubation), RobotML (incubation), UML-RT (incubation), and ISO 42010.

4.4.3 AF3-Fortiss (OS)

[http://af3.fortiss.org/]

AF3 is a tool to develop embedded systems using models from the requirements to the hardware architecture, passing by the design of the logical architecture, the deployment and the scheduling. AF3 provides features to support the user ensuring the quality of the system, formal analyses, synthesis methods, and space-exploration visualization.

4.4.4 Wolfram System Modeler

[http://www.wolfram.com/system-modeler/]

Wolfram System Modeler is a modeling and simulation environment for cyber-physical systems.

4.4.5 Modelica

[https://www.modelica.org/]

Modelica® is a nonproprietary, object-oriented, equation-based language to conveniently model complex physical systems containing, for example, the following: mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents. An interesting recent addition to the Modelica eco-system is the OMG standardization of a SysML profile[37]. This enables a bidirectional transformation between SysML and Modelica models, and supports the simulation of SysML models.

4.4.6 UML/SysML

[http://www.uml.org/], [http://www.omgsysml.org/]

The OMG's Unified Modeling Language™ (UML®) helps specify, visualize, and document models of software systems, including their structure and design. Using any one of the large number of UML-based tools on the market, the user can analyze application requirements and design a solution that meets them, representing the results using UML 2.0's 13 standard diagram types.

The OMG Systems Modeling Language (OMG SysML™) is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which is used to integrate with other engineering analysis models. SysML represents a subset of UML 2 with extensions needed to satisfy the requirements of the UML™ for Systems Engineering. SysML leverages the OMG XML Metadata Interchange (XMI®) to exchange modeling data between tools and is also intended to be compatible with the evolving ISO 10303-233 systems engineering data interchange standard.

SysML and UML are supported by commercial and OS tools. Sparx Enterprise Architect, IBM Rhapsody®, and Magic Draw® are commercial. Papyrus is an OS tool utilizing the Eclipse IDE framework.

Neither SysML or UML are modeling tools; they are containers within which a variety of documents, diagrams, and specifications can be held and shared between stakeholders. Custom parsers can be used to extract information and translate to other tools.

4.4.7 Ptolemy (OS)

[http://ptolemy.eecs.berkeley.edu/]

The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. A software system called Ptolemy II is being constructed in Java. The work is conducted in the Center for Hybrid and Embedded Software Systems (CHESS, http://chess.eecs.berkeley.edu/) in the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley.

5. TASK 1 VERIFICATION AND VALIDATION ISSUES

An often-cited root cause of accidents and incidents is that the original design requirements were in some way deficient, missing, or incomplete. Some examples of this are given in section 7. Design errors that were undetected because of improper verification were rarely at the root of accidents. This observation suggests that the current verification methods are operating adequately. It should be noted, however, that verification can verify only stated requirements and can neither identify incorrect ones nor reliably detect if some requirement is missing. Whereas it is observed that requirements errors predominate, this observation is not helpful unless one can postulate some way to overcome those deficiencies. A possible way to do this is through MBSE and MBSA combined with formal methods.

A two-step process is proposed:

1) Validation of requirements using MBSE.
2) FTA using MBSA to show compliance with current guidance. Both steps will be performed during the system-design phase.

5.1 REQUIREMENTS VALIDATION

Requirements creation in aerospace is currently performed under the guidance of ARP-4754A [38]. ARP-4761 [39] provides the supporting function of describing methods for developing safety requirements and showing that resulting design meets certain certification criteria detailed in the relevant advisory circular. The goal is to show that these processes need some strengthening to avoid accidents of the types related to requirements errors and omissions. The second aim is to suggest some rationale and methods to accomplish those improvements.

The central thesis is that specifying the functional and safety requirements of systems using a formal language allows the completeness (to the extent possible) and consistency of the requirements to be shown unequivocally using formal methods tools (model checker, fault tree generator).

A promising method that was investigated is creating formal requirements specifications using an assume-guarantee paradigm, otherwise known as contract-based design (CBD). Assume-guarantee is a form of formal requirements specification that can be checked for consistency and completeness using formal methods. A case study using this is given in section 6. CBD works by specifying for each block (component) of the architecture the assumptions that the block expects to be provided by the environment in which it lives. It also specifies a set of guarantees for what performance the block will provide in response if the environment provides what is assumed. These assume-guarantee formal specifications are expressed in LTL expressions that represent component behavior. They are therefore functional and safety specifications, and agnostic on hardware/software implementation details. This is in effect a property-based specification method of defining requirements, as outlined by Micouin [40]. This is repeated recursively for all blocks of the system-design hierarchy. In effect, the assume-guarantee pair provides the functional and safety requirements definition of each block, including the topmost system-level container block. System properties are proved compositionally at the top level by recursively discharging proofs of all the lower-level formal specifications, down to the leaf components.

The assume-guarantee pairs are used by a model checker to assure that all the block formal specifications are met by both itself and compositionally by its recursive child blocks. The model checker provides counterexamples in which they are not met as a trace of the localized port values to deduce the reasons for the miss and assist in identifying the cause. The counter examples therefore reveal where the formal requirements specifications are deficient.

CBD methodology is implemented in two tools: the AGREE AADL Annex [41] and the ESA COMPASS/FAME [20] tools. Underlying the ESA tools is a set of formal methods tools, System-Level Integrated Modeling (SLIM) language developed by FBK in Italy.

5.2 MODEL-BASED SAFETY ANALYSIS

The second step is MBSA FTA using the same MBSE functional models but extended by fault annotations of each leaf component block. Fault annotations specify the block failure mechanisms and modes. Leaf component failure modes and the associated occurrence probability become the basic events of the fault tree. Note that fault annotations are relevant only to random hardware failure because attempts to assign probabilities to software or human errors, omissions, or failure are ill conceived. Fault annotations are used to create a new composite system model via the extension of the functional models. In this way, low-level blocks, such as a sensor or actuator block, can have their failure modes and mechanisms specified so that they are reusable in other systems. Additional failure modes can be added, and the fault tree can be recreated relatively quickly.

Of note, MBSA is not strictly safety analysis; it is more akin to reliability analysis and the production of an FTA. In this regard, it follows the same philosophy as ARP-4761, in which safety analysis is reduced to reliability analysis. The reader is referred to section 5.3 for a fuller discussion of safety-analysis methods.

The hierarchy and structure defined in the top-level architecture model enable fault trees to be created for hazardous TLEs. The TLEs are defined using the same formal specification definition. The use of the STPA methodology [42] is suggested for identifying the system-level hazards. System hazards then become the TLEs of a fault tree. Hazards may or may not involve or require a failure. Safety and reliability are not the same, so a poor system design may be hazardous irrespective of failure. Some examples for which this is the case are given in (see section 6). Conversely, low reliability does not equate to hazardous, although it will adversely affect availability. The system design must be modified to eliminate or mitigate the identified hazards to the level required by the certification guidance. If a failure is necessary for the hazard to occur, the fault tree will compute the TLE probability. This probability is then assessed against a certification requirement, such as a design assurance level (DAL). If the probability is acceptable per the certification guidance, no further action is needed. If not, the hazard must be mitigated with a design change. In this way, the need for redundancy, fault tolerance, or better component reliability can be assessed. Showing achievement of some designated DAL does not of itself provide any guarantee or assurance of safety. It does, however, provide confidence that the design team has followed a prescribed process that has historically performed well.

This process naturally leads to successive iterations of the system and components during the design phase until all identified hazards are eliminated or mitigated to the required level. This is all carried out within a virtual integration environment and prior to the build of prototype hardware.

The key point to emphasize is that safety analysis in general, and MBSA in particular, requires both the functional behavior and the fault behavior to be integrated into a single modeling and hazard-analysis process. Safety is a system property like any other, so its assess5.ment is inextricably connected to the system design and cannot be treated as an after-the-fact exercise.

The AADL AGREE (section 4.1.19) and BLESS (see section 4.1.18) Annex specifications are purely functional and, therefore, do not support fault-insertion behavior. Both are disjointed from fault tree tools, such as the AADL EMV2 Error Annex. Similarly, HiP-HOPS (see section 4.1.2) has no support for system or block functional behavior. Both these FTA tools require the analyst to compose the fault tree in the form of composite error behavior Boolean functions. This requires the analyst to manually construct the fault tree equations for a given TLE to capture the effect of a fault on functional behavior. This does not lend itself to effective missing or incomplete requirements discovery because counter examples are not readily uncovered. Because system safety cannot be separated from system function, it is considered essential to closely integrate MBSE and MBSA into one process operating on the same model.

A further advantage of the combining functional and reliability analysis is the ability to consider multiple faults. This is referred to as cardinality. Using manual methods of composite-error construction for single faults (cardinality 1) is conceivably possible, but the complexity of two or more concurrent faults makes this an infeasible approach, except in the simplest of systems. Multiple concurrent faults can be handled automatically by the CBD method. At this stage, the constraints and limitations imposed by the practical limitations of computing platforms on the scalability of the approach in both the complexity and cardinality dimensions are not fully understood. Further work on aviation examples with representative complexity is needed.

The COMPASS and FAME tools support both the above MBSE and MBSA methodologies. However, these are not licensed for use outside of Europe. These tools utilize a variant of the AADL language called SLIM to express models and a graphical front end for system design is provided. The underlying technologies for both COMPASS and FAME is a model checker and a fault tree analyzer provided by FBK in the OCRA toolset (section 4.1.10). The OCRA tools do not use the SLIM language, they use their own dialect, and neither is a GUI provided to capture the system.

A case study (section 6) shows how a UML/SysML tool (e.g., Papyrus) can be used to capture the architecture block structure and connectivity, the assume-guarantee formal specifications and signal type specifications in a GUI. The tool interfaces to OCRA via a translation of the underlying XML file. XML is translated to the OCRA language in a few seconds using a custom piece of software developed partly under this program. A few lines of shell script processes the OCRA file through the model checker and fault tree analyzer.

The OCRA tool has been demonstrated by FBK as applied to the SAE AIR-6110 wheel brake system example, and full results are available in [43]. The limitations of OCRA are more fully explored in section 4.1.10. These results show that compute time increases exponentially with cardinality to the point that analyses of high cardinality cannot always be accomplished in reasonable time or with readily available engineering workstations.

The assume-guarantee formal requirements specification approach nevertheless appears to be promising, notwithstanding the OCRA limitations. It provides a partial solution from the initial block diagram architecture of an aircraft function captured in a way familiar to most systems engineers, to a validated set of block-level requirements given with well-defined semantics. It is extensible to almost arbitrary levels of model scope, size, and fidelity, subject to yet unknown limitations of practical computational platform capability.

5.3 SAFETY ANALYSIS METHODS

There are hazard-analysis methods that identify the cause of hazards (which can lead to accidents), and there are risk-analysis methods that attempt to associate a risk metric (combination of severity and likelihood) with identified hazards. Both are described in section 5.3.1 with the tools used for them.

5.3.1 Hazard Analysis Methods

Hazards are states of the system that, combined with worst-case environmental conditions, can lead to an accident (loss). Hazard analysis is the process of identifying scenarios (arising from the system design) that can lead to system hazards (hazardous system states). Note that hazards are system states or system conditions; they are not errors or failures (which can be some of the causes of hazards but not necessarily the only causes).

Hazard analysis depends on assumptions about why hazards (and therefore accidents) occur. Traditionally, accidents have been assumed to result from component failures and faults (i.e., they stem from unreliable system components). Increasing the reliability of the components and designing to protect against failures of individual components should eliminate accidents. This assumption was reasonable 50 years ago (when current hazard- and accident-analysis methods were first created) because systems were relatively simple compared to today and consisted only of basic hardware components. Software began to be introduced into engineered systems approximately 30 years ago. In the relatively simple systems built before this time, design errors could be eliminated during development by reviewing designs carefully and through relatively exhaustive testing. What remained to cause problems during operations were hardware failures and operator errors in following prescribed procedures.

In contrast, systems today are increasingly becoming software intensive. Software is allowing much more complex systems to be designed while making the removal of design errors through exhaustive testing impossible. The complexity is also making it difficult for engineers to identify all the different ways the system components can interact and therefore predict and understand all potential system behavior. The result is system-design errors remaining during operations and leading to accidents.

In addition, the complexity of modern systems is making them more difficult to operate, which is changing operator errors. Operators are also having trouble understanding and predicting how these software-intensive systems will behave. The system design (including the software functional design) is actually inducing new types of operator errors.

These changes are making traditional hazard and accident-analysis methods, such as FTA, Hazard and Operability Analysis (HAZOP), event trees, and Failure Modes Effects and Criticality Analysis increasingly less effective. Leveson has argued that something new is needed that fits engineering today and not the systems before computer [44]. The assumptions about the cause of accidents that underlie the traditional hazard methods may not be sufficient to mitigate the risks of today's software-intensive systems.

Similarly, because software and humans are usually excluded from the hazard analysis performed under SAE ARP-4761[39], risks related to software and human failures may be inadequately explored. Instead, in the current practice, humans are not considered (except as mitigators of hazards, not causers of hazards). Similarly, software failure is not directly addressed within the current development processes. Instead, software is assigned a DAL, in accordance with its potential contribution to the functional hazard assessment. The DAL is then used to determine different levels (A through E) of rigorous development processes, which need to be performed as part of the software implementation. It should also be noted that the prescriptive guidelines of DO-178B/C have been very successful with respect to reducing software implementation bugs. However, design assurance alone does not ensure safety. The larger problem is that virtually all software-related accidents in aviation and other fields have resulted from inadequate software requirements and not from errors in the implementation of the requirements in the software.

5.3.2 Risk Analysis (Risk Assessment)

Risk analysis is the process of identifying the severity and likelihood of hazards occurring. The use of likelihood assumes there is some type of randomness or a stochastic process involved. This assumption does not hold for software, which is pure design abstracted from its physical realization [45]. In practice, software is either ignored in the risk analysis (as with SAE ARP-4761) or incorrect assumptions are made—for example, that software always has a "failure" probability of 10^{-4} (using whatever units are appropriate for the problem).

Risk analysis can be performed in a semi-qualitative way in which likelihood is assigned to broad categories, such as "highly likely, likely, unlikely, or impossible." Alternatively, quantitative methods can be used. The latter is usually called PRA. Common assumptions about independence of failures may make PRA results unrealistic. An advantage of PRA is that standard hazard-analysis techniques, such as FTA, can be used to calculate the risk. The problem, of course, is that many of the causes of accidents in today's complex, software-intensive systems are omitted from traditional hazard-analysis techniques and therefore cannot be included in the PRA in any meaningful fashion. As a result, the results of PRA may be misleading.

5.3.3 System Theoretic Accident Model and Process (STAMP)

STAMP is a new accident causality model, based on systems theory rather than reliability theory. STAMP extends the traditional assumptions about accident causality to include not only component failure and faults but system design errors and unplanned, unanticipated interactions among components that have not failed (i.e., operate exactly as they were designed to work). Because software does not fail, the latter is the usual mode in which software contributes to accidents. The problem almost always is in the requirements and not in coding errors. SAE ARP-4761 ignores this fact and concentrates on the rigor of the process used to implement the requirements.

In STAMP, accidents result from a large variety of causes, including component failures and faults, system-design errors, unintended and unplanned interactions among components, operator errors, flawed management decision making, inadequate system and software requirements, and poor safety culture. Analysis methods built on STAMP can identify potential hazards resulting from any of these causal factors. STAMP is based on control theory rather than reliability theory and treats accidents as arising from inadequate control over the behavior of the system components and not just component failures or errors. Because STAMP uses the same theoretical foundation as systems engineering, STAMP-based analysis methods can easily be integrated into system-engineering processes and used to design safety into the system as design decisions are being made.

Systems theory has four basic concepts: hierarchy, emergence, communication, and control.

1. Hierarchy: A general model of complex systems can be expressed in terms of a hierarchy of levels of organization, each more complex than the one below.
2. Emergence: Each level of the hierarchy is characterized by having emergent properties. These properties do not exist at lower levels but arise from the operation of the processes at a lower level of the hierarchy. Safety and security are examples of emergent properties but are not the only ones usually of interest in a system. Emergent properties associated with the interactions of the set of components at one level in a hierarchy are related to constraints on the degree of freedom of those components. Two examples of high-level safety constraints are that propulsion sufficient to maintain lift must be available when the aircraft is airborne and that braking (deceleration) capability must be adequate to stop the aircraft's forward progress on landing in the landing space available.
3. Control: Control involves the imposition of constraints on the activity at a lower level of the hierarchy (i.e., at the interfaces between levels). The imposition of safety constraints on the behavior of the system components plays a fundamental role in a systems approach to safety.
4. Communication: Control implies the need for communication between levels of the hierarchy and between the components at each level.

Figure 3 shows the basic form of a feedback-control structure in which the controller imposes control actions on the controlled process. Such control loops can be composed into more complex control structures (a hierarchy of control).

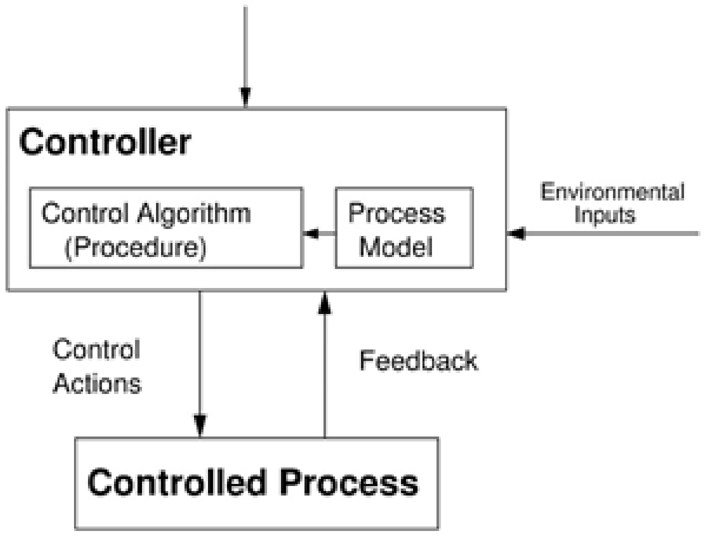


Figure 3. The basic building block for a safety-control structure

Figure 4 shows an example of a hierarchical safety-control structure for a typical aircraft SMS.

Feedback control uses information about the current state of the controlled process (usually derived at least partially from feedback). In STAMP, this information is called the process model. For humans, this concept is part of what is often called a mental model. Note that the types of mental model flaws relate to human factors concepts, such as mode confusion or situation-awareness errors. One example is that the pilot does not think that the aircraft is in a stall condition (when it really is), and the pilot provides unsafe control actions as a result.

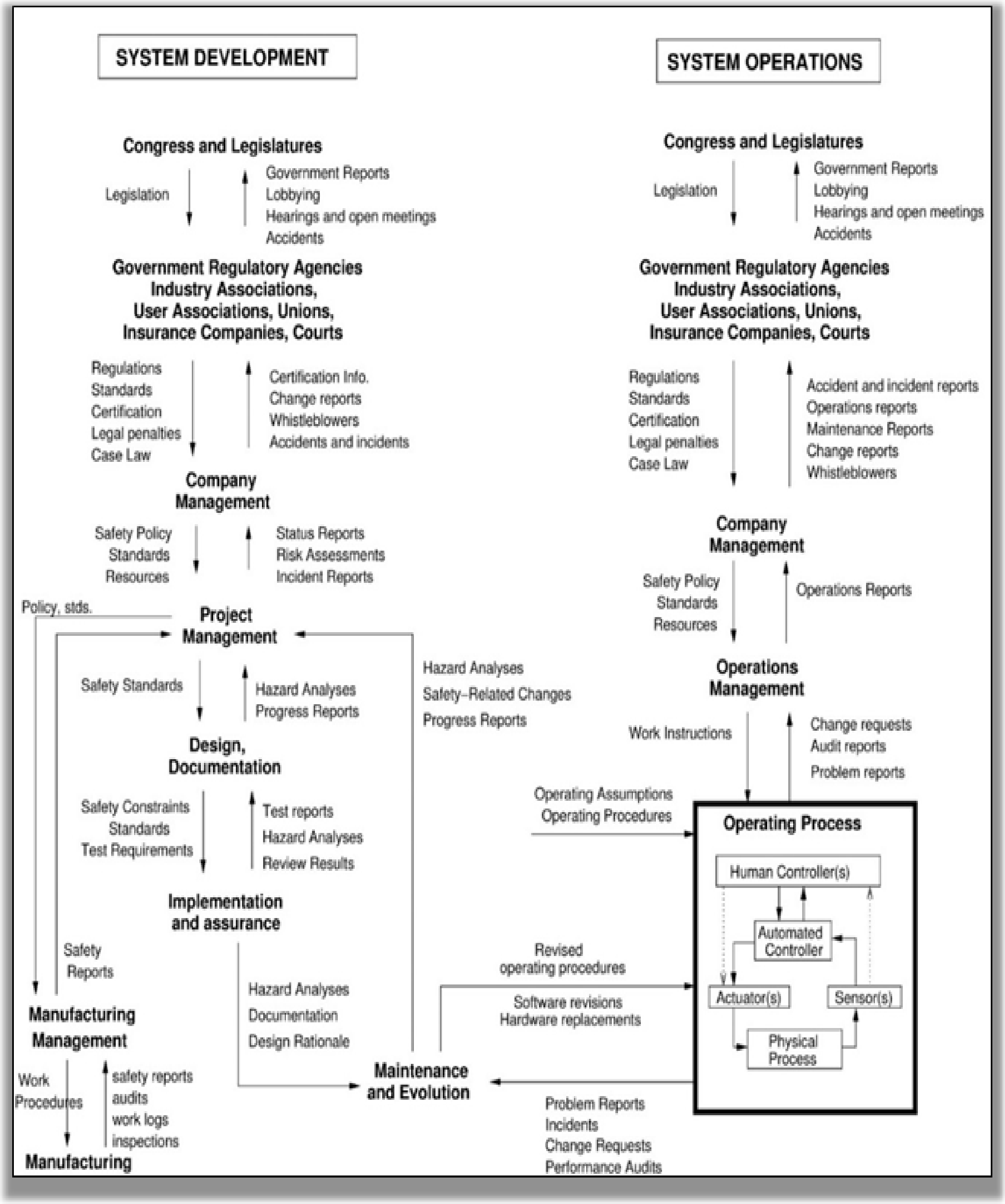


Figure 4. Example safety control structure

STAMP also provides a better conception for how software contributes to accidents. Software controllers always contain a process model, although it often is not called that and may consist of a few variables that represent the assumed state of the controlled process. Those variables are used to determine what control actions (outputs) are provided by the software. Accidents often result when the process model becomes inconsistent with the actual state of the process, and the controller provides an unsafe control action. The software does not know the aircraft has landed and prevents the pilot from activating the reverse thrusters (a constraint added for safety reasons to prevent the reverse thrusters from activating while the aircraft is airborne), resulting in the type of accident that occurred in Warsaw and other instances. The STAMP model of accident causation is much more appropriate for understanding and mitigating accidents arising from human and automated control systems (software) than a model that assumes accidents result from random failure behavior. Humans and software do not simply randomly fail like hardware.

Process models are kept up to date through feedback. A common cause of accidents is that appropriate feedback is incorrect, missing, or delayed. There are four types of unsafe control actions:

- A provided control action leads to a hazard.
- A control action not provided leads to a hazard.
- A control action provided with wrong timing (early, late) or order leads to a hazard.
- A continuous control action provided too long or too short a time leads to a hazard.

These four types of unsafe control actions, with the hierarchical safety-control structure, are used for the hazard analysis.

STPA is an analysis technique based on STAMP that can be used to analyze a system's functional-control structure for emergent properties. When used for safety, it becomes a hazard-analysis technique. It can also be used to analyze a functional control structure for cyber security vulnerabilities. CAST is another analysis process based on STAMP that can be used to perform a causal analysis after an accident or incident to more completely identify all the factors involved as compared to classic root cause analysis techniques. CAST can identify the causal scenario for an accident that has already occurred. STPA and STPA-Sec are used for proactive hazard and security analysis (i.e., identifying the scenarios that could lead to a loss in the future). This information is used to protect the system from such scenarios, either by eliminating them in the system design or mitigating their occurrence if they cannot be eliminated.

5.3.4 Specification Tools and Requirements Methodology (SpecTRM) and SpecTRM-RL

During the certification of Traffic Collision and Alerting System (TCAS) II in the early 1990s, a model-based specification language was created to assist in the certification effort. Some requirements were that the language had to be learnable and readable with minimal effort (in less than an hour) by anyone who wanted to review the specification, including non-engineers[46].

Lessons learned from this effort were used to create a commercial requirements management tool called Specification Tools and Requirements Methodology (SpecTRM) [47]. The formal, model-based specification black-box Requirements State Machine Language, used for specifying and certifying TCAS II, was improved based on lessons learned from that effort and named SpecTRM-RL. However, much more is needed in system engineering to document and develop requirements specifications than simply a formal, model-based language. To achieve that goal, a complete specification methodology (Intent Specifications) was developed and formed the basis for the larger SpecTRM toolset.

Much of the information needed by engineers to successfully create complex systems is best documented in English-language statements, is organized in a fashion that includes the specification of underlying assumptions and rationale, promotes traceability (in both directions), and augments the ability to find the information necessary to make decisions when needed. SpecTRM supports traditional requirements and design specifications but is structured in a slightly different fashion to make them more usable and effective[48].

In addition, SpecTRM supports a black-box, functional, model-based specification of the system built on a formal methods foundation. The formal specification component is called SpecTRM-RL. The SpecTRM-RL notation itself is tabular and graphical, and can be easily read and created by engineers without extensive mathematical background [46]. The formal foundation of the language, however, allows the specifications to be executed and analyzed so that errors can be found by automated tool support. Existing SpecTRM tools include both static and dynamic analysis tools, completeness analysis, consistency analysis, mode-confusion analysis, test-data coverage and generation tools, and alternative ways to display the information to assist in understanding and answering specific questions about the functional requirements. The simulation tools allow for animation and visualization of the specification and integration with other models such as Simulink. The alternative display format tools have been validated using controlled human experiments to support the claim that they improve human ability to understand and change specifications without introducing errors [44]. SpecTRM is built on an Eclipse platform and, therefore, is easily combined with other tools. It has been used in industry for the development of complex systems for approximately 20 years, mostly in the military and space domains.

SpecTRM embodies the concepts of Intent specifications [48], which were designed based on systems theory, systems engineering principles, and psychological research on human problem solving and how to enhance it. The goal is to assist humans in dealing with complexity. An Intent specification differs from standard specifications primarily in its structure rather than the information included. The information is organized in a way that has been found to assist in its location and use. These hypotheses have been validated in carefully designed and controlled human laboratory experiments [49].

Most complex systems have voluminous documentation, much of it redundant or inconsistent, and it degrades quickly as changes are made over time. Intent specifications were designed based on systems theory, systems engineering principles, and psychological research on human problem solving and how to enhance it. The goal of intent specifications is to organize the system engineering information to improve the usefulness of the specifications. Sometimes important information is missing, particularly information about why something was done the way it was (i.e., the intent or design rationale). Trying to determine whether a change might have a negative impact on safety, if possible at all, is usually enormously expensive and often involves regenerating analyses and work that was already completed but either not recorded or not easily located when needed. Design rationale, safety-analysis results, and the assumptions on which the system and its validation are based are integrated directly into the system specification and its structure rather than being stored in separate documents, so the information is at hand when needed for decision making.

The structure of an intent specification is based on the fundamental concept of hierarchy in systems theory, in which complex systems are modeled in terms of a hierarchy of levels of organization, each level imposing constraints on the degree of freedom of the components at the lower level. Each level (model) also provides a different, but complete, view of the system.

Figure 5 shows the structure of an intent specification. It is organized along three dimensions: intent abstraction, part-whole abstraction, and refinement. These dimensions constitute the problem space within which the user navigates. Part-whole abstraction (along the horizontal dimension) and refinement (within each level) allow users to change their focus of attention to detailed views within each level or model. The vertical dimension specifies the level of intent or "why" at which the problem is being considered.

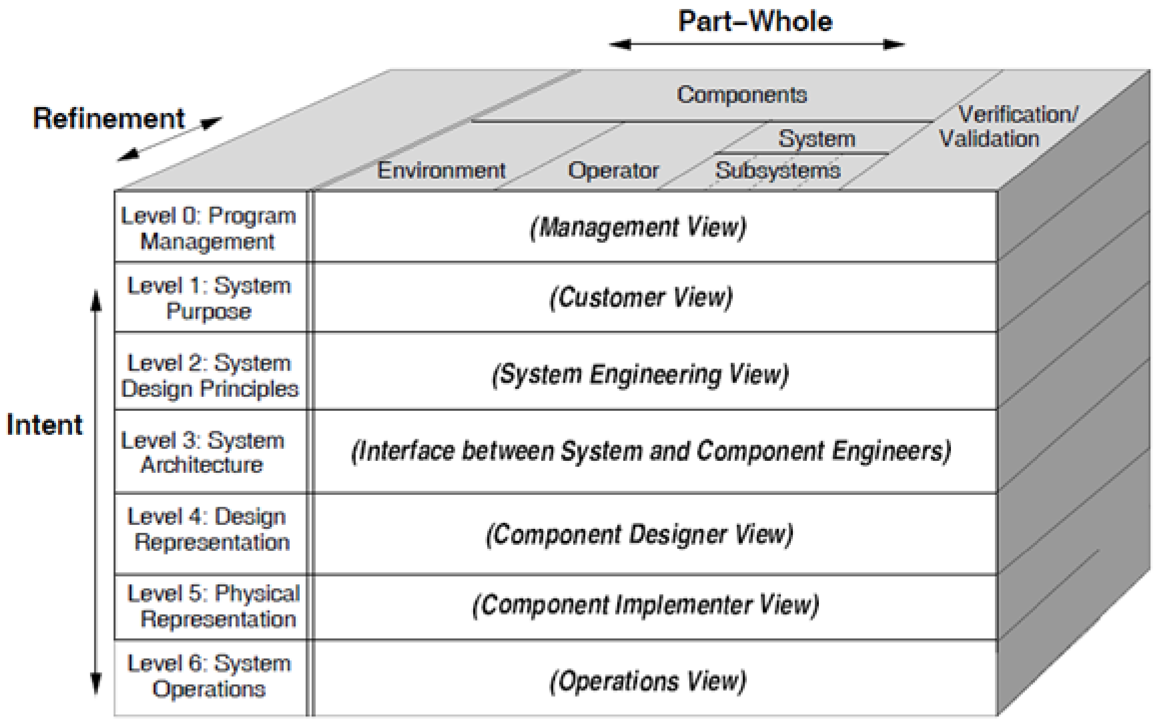
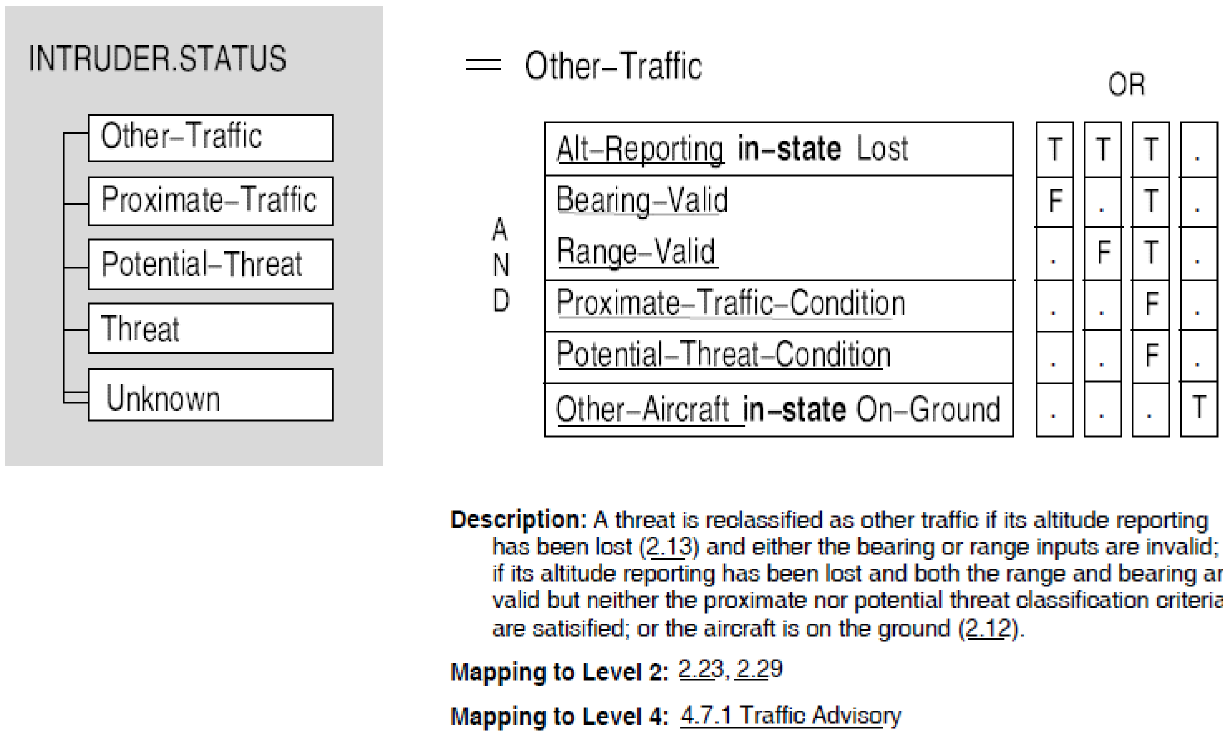

Part–Whole
Refinement
Components
Environment
Operator
System
Subsystems
Verification/ Validation
Intent
Level 0: Program Management
(Management View)
Level 1: System Purpose
(Customer View)
Level 2: System Design Principles
(System Engineering View)
Level 3: System Architecture
(Interface between System and Component Engineers)
Level 4: Design Representation
(Component Designer View)
Level 5: Physical Representation
(Component Implementer View)
Level 6: System Operations
(Operations View)

Figure 5. Structure of an intent specification

Level 3 is the only level at which formal models were found to be useful. This level specifies the system architecture and serves as an unambiguous interface between system engineers and component engineers or contractors. SpecTRM-RL is used here to decompose the system functions, allocate them to components, and specify them rigorously and completely. Black box SpecTRM-RL models are used to specify and reason about the logical design of the system and the interactions among individual system components without being distracted by implementation details.

SpecTRM-RL models are based on an underlying state machine model, but use tables to specify behavior. In formal experimentation and empirical usage (during the large number of TCAS reviews by aerospace engineers and pilots), it was found to be the best way to make formal specifications readable and easily learnable [46].

Figure 6 shows an example from the TCAS II Level-3 specification. This defines the criteria for downgrading the status of an intruder (into the protected volume) from being labeled a threat to being considered simply as other traffic. Intruders can be classified in decreasing order of importance as a threat, proximate traffic, and other traffic. In the example, the criteria for taking the transition from state "Threat" to state "Other Traffic" is represented by an AND/OR table, which evaluates to TRUE if any of its columns evaluate to TRUE. A column is TRUE if all its rows that have a "T" are TRUE, and all rows with an "F" are FALSE. Rows containing a "." are "don't care" conditions. These tables capture predicate logic formulas.



Alt–Reporting **in–state** Lost	T	T	T	.
Bearing–Valid	F	.	T	.
Range–Valid	.	F	T	.
Proximate–Traffic–Condition	.	.	F	.
Potential–Threat–Condition	.	.	F	.
Other–Aircraft **in–state** On–Ground	.	.	.	T

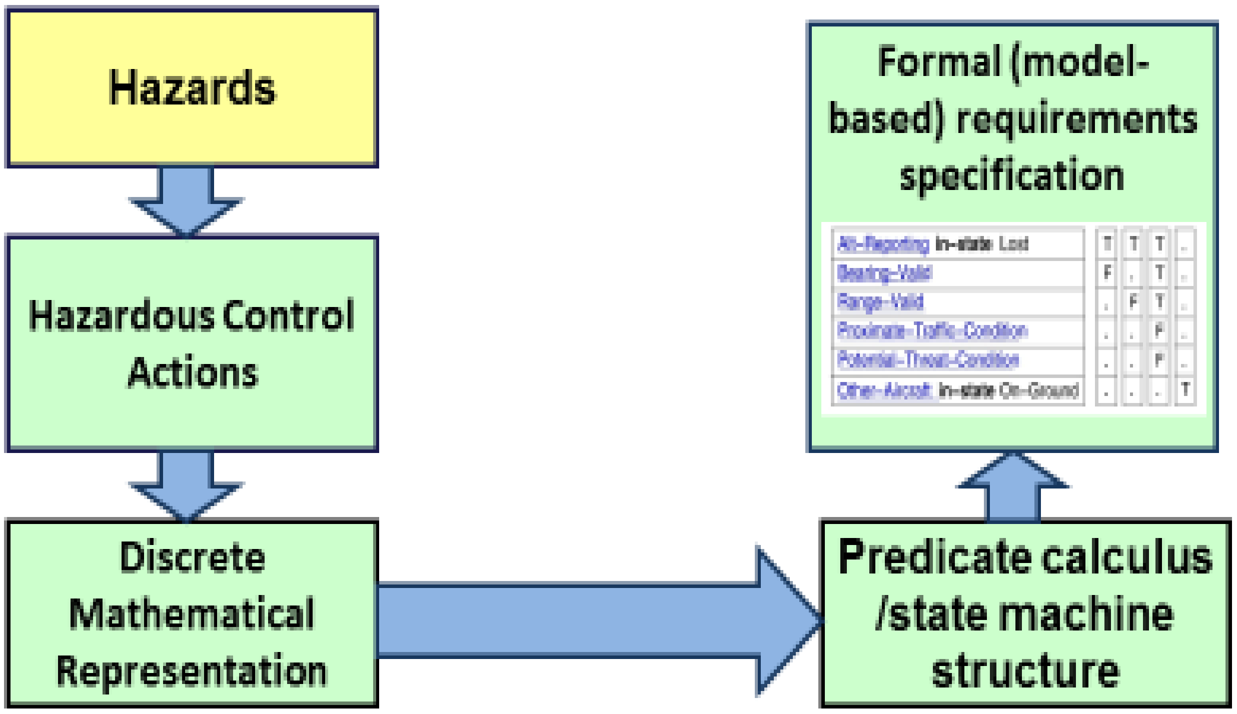
Description: A threat is reclassified as other traffic if its altitude reporting has been lost (2.13) and either the bearing or range inputs are invalid; if its altitude reporting has been lost and both the range and bearing are valid but neither the proximate nor potential threat classification criteria are satisified; or the aircraft is on the ground (2.12).

Mapping to Level 2: 2.23, 2.29

Mapping to Level 4: 4.7.1 Traffic Advisory

Figure 6. TCAS level 3 SpecTRM-RL model

STAMP and STPA are being integrated into the SpecTRM toolset. Thomas [8] has shown how, starting from an English language specification of the hazards (which would be at levels 1 and 2 of the intent specification), tools based on formal methods can be used to automate the generation of the SpecTRM-RL tables (see figure 7) and to identify inconsistencies and conflicts.


Hazards
Hazardous Control Actions
Discrete Mathematical Representation
Predicate calculus /state machine structure
Formal (model-based) requirements specification

Alt-Reporting In-state Lost	T	T	T	.
Bearing-Valid	F	.	T	.
Range-Valid	.	F	T	.
Proximate-Traffic-Condition	.	.	F	.
Potential-Threat-Condition	.	.	F	.
Other-Aircraft In-state On-Ground	.	.	.	T

Figure 7. Generating SpecTRM-RL tables

Tools have been demonstrated to fully automate, or in some cases assist, human engineers in creating SpecTRM-RL specifications of the system and software safety requirements given a basic list of system hazards.

6. FORMAL METHODS CASE STUDY

To show the process of MBSA using formal requirements specifications, a worked example of its application to a notional aircraft wheel-braking system is shown. The system selected is representative of a real system but does not contain proprietary information. The example is similar to that described in AIR-6110 [50], from which the hazard analysis is also used.

The system capture in SysML, the formal requirements specifications attached to each block of the system, the automatic model translation from SysML to the text format system specification used by the analysis tool and, finally, the results from the application of the analysis tools are shown. A summary is presented here to show the main results. The full details with all necessary scripts are more fully expanded in [31] and [51].

The toolset adopted is:

- FBK OCRA [52], NuXMV[29] model checker and xSAP[34] MBSA tools
- Open source Papyrus SysML model editor
- Open source Prolog (SWI) and Python interpreters (an internal variant of the LMP tooling approach described in section 10.1)

For the reasons explained in section 4.1, this toolset offers the best integration of functional and safety requirements specification methods combined with an integrated method for implementing the processes of ARP-4761.

6.1 WBS ARCHITECTURE

The top-level architecture is shown in figure 8. The aircraft is fitted with four wheels on each side and has pedal position sensors for the pilot and copilot. The top-level architecture is initially divided into the control system (the electrical units) and the physical system (the hydraulic system of valves, pipes, pumps, and brakes). The design is hierarchical so that these subsystems break down into smaller subsystems and eventually into leaf components, such as electronics units, valves, and brakes (see figure 9).

To capture the design, the Papyrus SysML tool was used. This tool supports the capture of hierarchical designs in block diagram form. The output from this tool is an XML file representing the entire hierarchy.

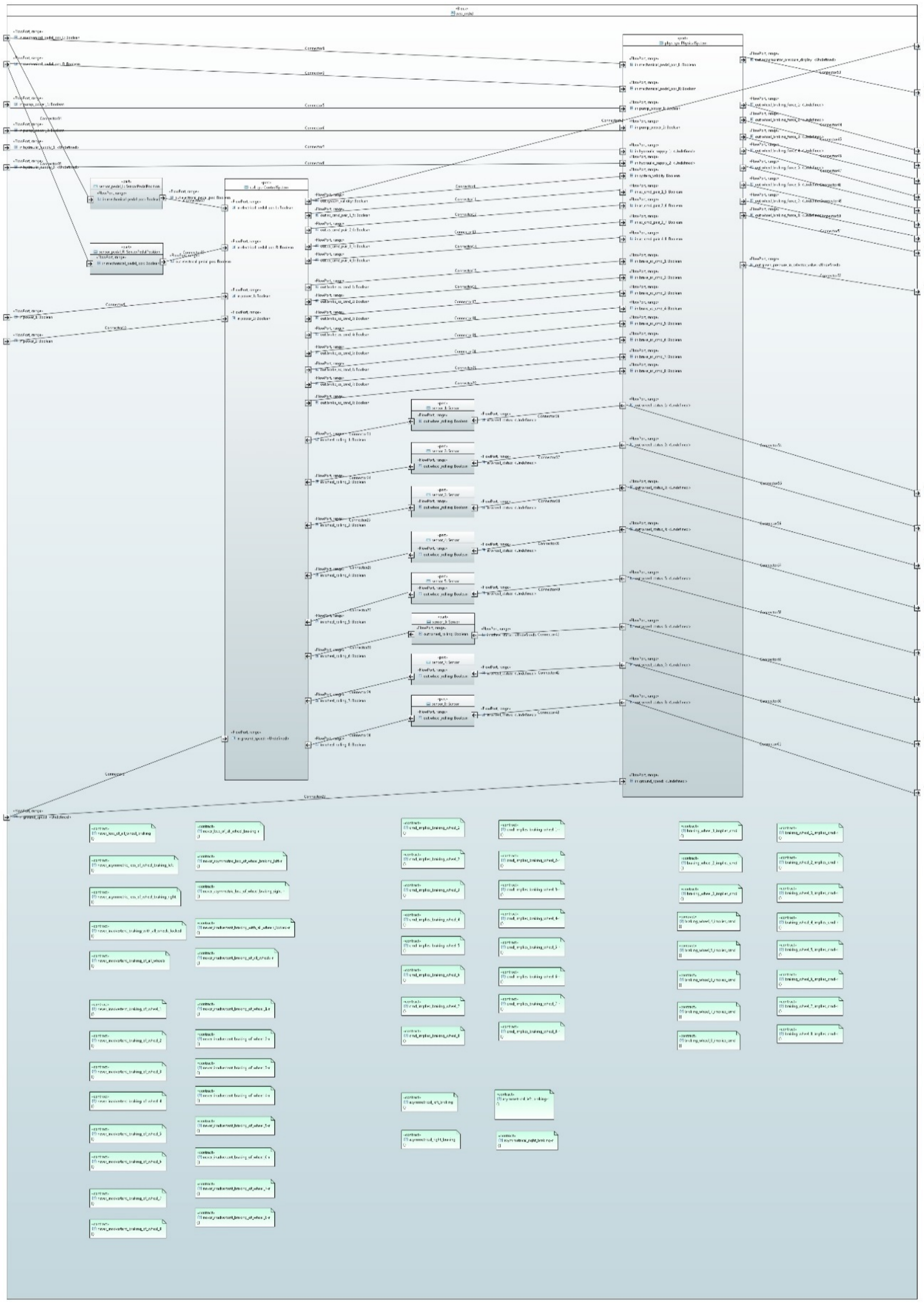


Figure 8. Top-level WBS architecture

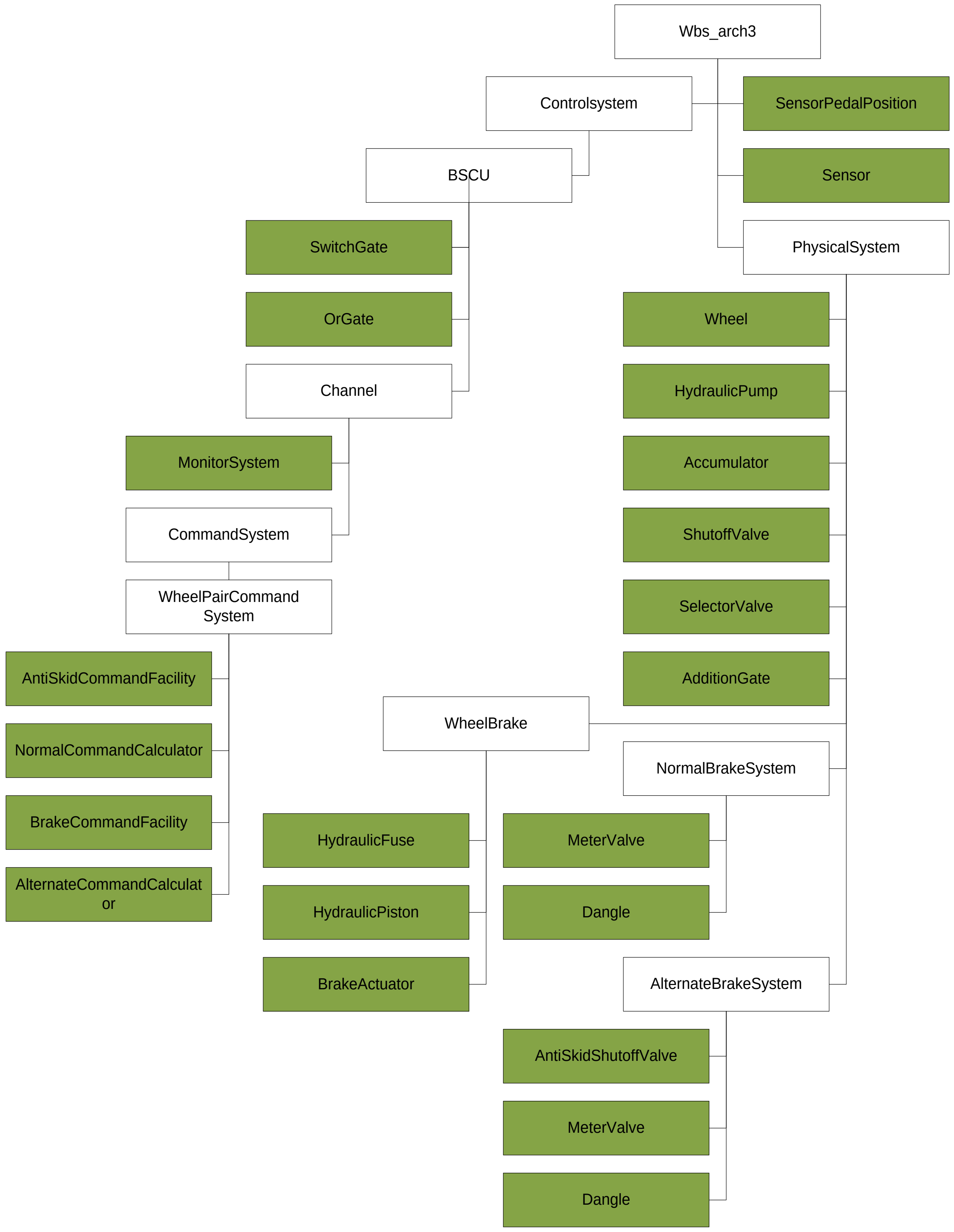
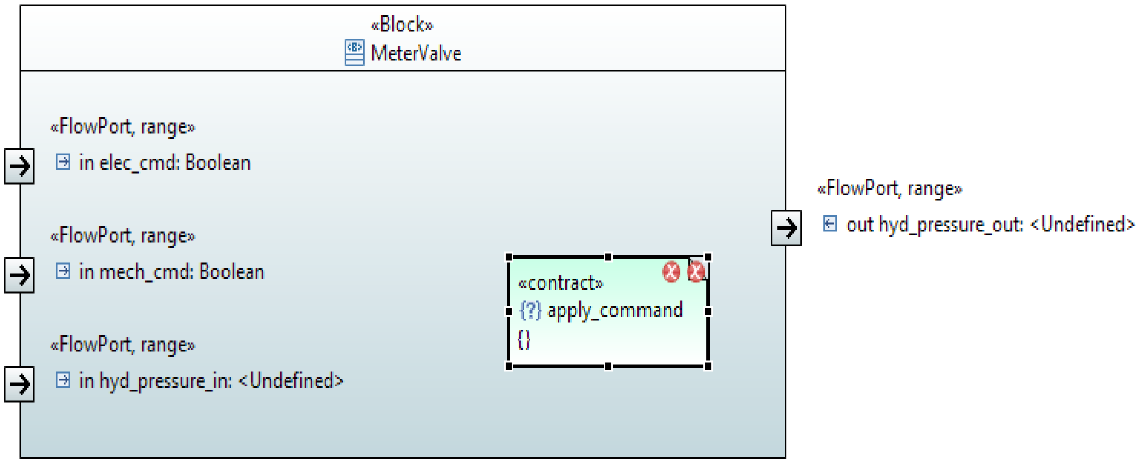
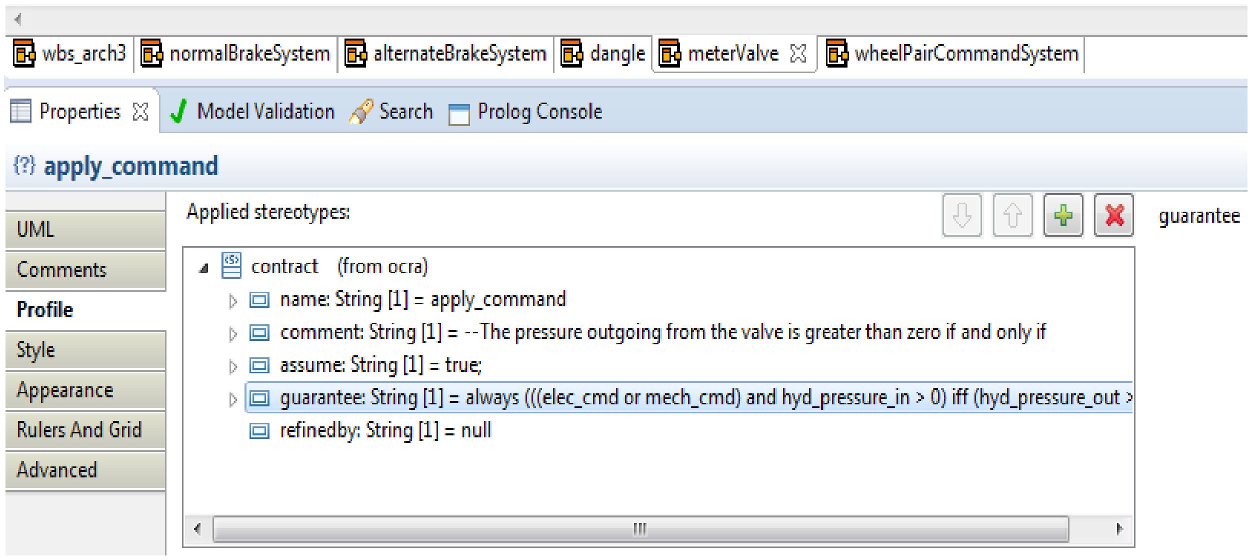


Figure 9. WBS hierarchy (leaf components in green)

Each system block contains zero or more formal requirements specifications (or contracts). As a simple example of a block definition, shown below is a leaf block (the meter valve) containing a formal requirements specification (see figure 10). Each formal requirement comprises an assumption and a guarantee, as shown in the lower part of the figure.





{?} apply_command

Applied stereotypes: guarantee

```
contract    (from ocra)
    name: String [1] = apply_command
    comment: String [1] = --The pressure outgoing from the valve is greater than zero if and only if
    assume: String [1] = true;
    guarantee: String [1] = always (((elec_cmd or mech_cmd) and hyd_pressure_in > 0) iff (hyd_pressure_out >
    refinedby: String [1] = null
```

Figure 10. Meter valve block diagram

The full expansion of the guarantee part of the specification is:

```
always (((elec_cmd or mech_cmd) and hyd_pressure_in > 0) iff
   (hyd_pressure_out > 0));
```

This is a logical expression in the OCRA language. This example is interpreted to mean that the logical expression of signals during fault-free operation is expected to always be true (i.e., the hydraulic pressure output is provided if either the electrical or mechanical command is provided and there is a valid supply of hydraulic pressure. This block has inputs for the hydraulic pressure feed, and electrical and mechanical actuation inputs and an output of hydraulic pressure to the wheel brake. The formal specification of the block function is captured in SysML as a UML profile of type “contract” with a specific instance name of “apply_command.”

6.2 MODEL TRANSFORMATION

The next step in the process is to translate the XML format created by the SysML tool into the OSS text format used by OCRA. A two-step process was implemented to do this. In the first step, the SysML format (XML) was translated into a Prolog fact base using a custom translator tool written in Python. This fact base is then processed by a custom Prolog program that constructs the OSS text file from the Prolog fact base. The resulting fragment of the OSS file for the meter valve is:

```
COMPONENT MeterValve
   INTERFACE
      INPUT PORT elec_cmd : boolean ;
      INPUT PORT mech_cmd : boolean ;
      INPUT PORT hyd_pressure_in : 0..10 ;
      OUTPUT PORT hyd_pressure_out : 0..10 ;
   CONTRACT apply_command
   --The pressure outgoing from the valve is greater than zero if
   and only if
   -- an hydraulic pressure is present in input
   -- and an electrical brake command or a mechanical brake command
   is received
   assume: true;
   guarantee: always (((elec_cmd or mech_cmd) and hyd_pressure_in
   > 0) iff (hyd_pressure_out > 0));
```

A more complex example is shown in figure 11. This block utilizes four AntiSkidShutofValves and four MeterValves. It also has four pairs of linked formal requirements specifications attached; one pair for each of the four outputs. Each pair consists of a functional specification for the required operation of the associated output and a reference to the formal specification of the contained block. This process of linking to lower level specifications is called refinement (using the REFINEDBY keyword) and is accomplished by name matching of specifications—in this case, apply_command_1.

```
CONTRACT apply_command_1
--The pressure outgoing from each port of the system is greater
   than zero if and only if:
-- a hydraulic pressure is present in input and the assigned
   mechanical pedal is pressed and there is no anti-skid command
   for the pairs of wheels
   assume: true;
   guarantee:   always   (((mechanical_pedal_pos_L   and   not
   as_cmd_pair_1_5       and       hyd_pressure_in>0)       iff
   (hyd_pressure_out_1>0)));
CONTRACT apply_command_1
--subcontract
   REFINEDBY
      meter_valve_1.apply_command,
   antiskid_shutoff_valve_1.apply_command;
```

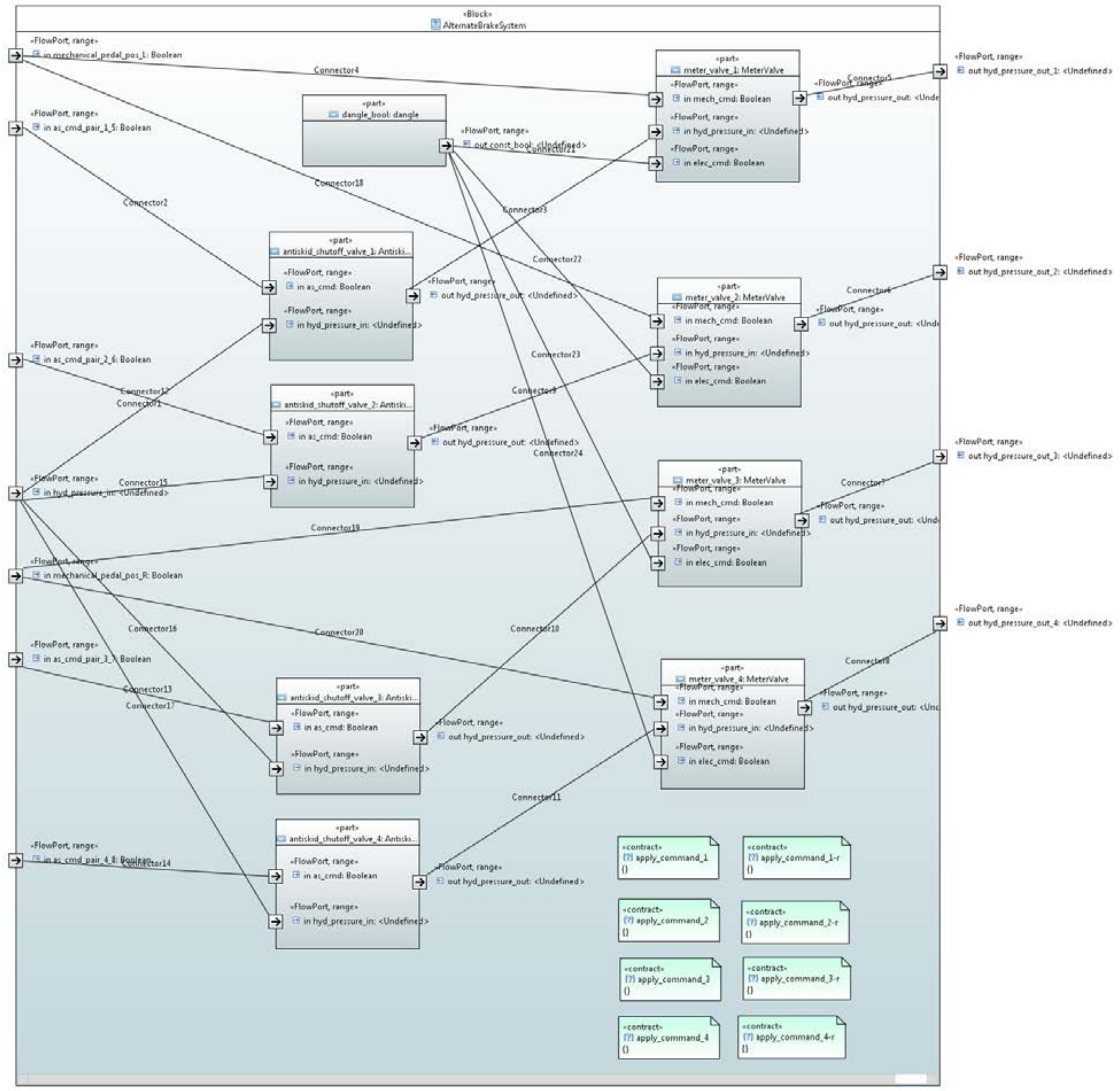

«Block»
AlternateBrakeSystem
«FlowPort, range»
in mechanical_pedal_pos_L: Boolean
in as_cmd_pair_1_5: Boolean
in as_cmd_pair_2_6: Boolean
in hyd_pressure_in: <Undefined>
in mechanical_pedal_pos_R: Boolean
in as_cmd_pair_3_7: Boolean
in as_cmd_pair_4_8: Boolean
«part»
dangle_bool: dangle
out const_bool: <Undefined>
antiskid_shutoff_valve_1: Antiski...
antiskid_shutoff_valve_2: Antiski...
antiskid_shutoff_valve_3: Antiski...
antiskid_shutoff_valve_4: Antiski...
in as_cmd: Boolean
out hyd_pressure_out: <Undefined>
meter_valve_1: MeterValve
meter_valve_2: MeterValve
meter_valve_3: MeterValve
meter_valve_4: MeterValve
in mech_cmd: Boolean
in elec_cmd: Boolean
out hyd_pressure_out_1: <Undefined>
out hyd_pressure_out_2: <Undefined>
out hyd_pressure_out_3: <Undefined>
out hyd_pressure_out_4: <Undefined>
«contract»
apply_command_1
apply_command_1-r
apply_command_2
apply_command_2-r
apply_command_3
apply_command_3-r
apply_command_4
apply_command_4-r

Figure 11. AlternateBrakeSystem block diagram

At the system level, the hazards that are to be avoided are defined. These would in general be derived from the Safety Assessment process (see [39] and [84]), but the hazards defined in AIR-6110 are used here. As an example, consider the system hazard "never_loss_of_all_wheel_braking" [50]). This is expressed in a formal specification again using the assume/guarantee formalism. The assumption is that electrical and hydraulic power sources are available and, given that this assumption is met, the implementation will meet the specification provided in the guarantee.

```
CONTRACT never_loss_of_all_wheel_braking
-- S18-WBS-R-0321 p 58
-- Loss of all wheel braking (annunciated or unannunciated) during
   landing or RTO shall be extremely remote
   assume: always (power_1 and power_2 and pump_power_1 and
   pump_power_2 and hydraulic_supply_1=10 and
   hydraulic_supply_2=10);
   guarantee: never ((mechanical_pedal_pos_L and
   mechanical_pedal_pos_R) and ground_speed>0 and (not
   ((mechanical_pedal_pos_L and (((wheel_status_1=rolling or
   ground_speed=0) and green_pressure_in_selector_valve>0) or
   (((wheel_status_1=rolling and wheel_status_5=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_1>0) and not ((mechanical_pedal_pos_L and
   (((wheel_status_2=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_2=rolling and wheel_status_6=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_2>0) and not ((mechanical_pedal_pos_L and
   (((wheel_status_5=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_5=rolling and wheel_status_1=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_5>0) and not ((mechanical_pedal_pos_L and
   (((wheel_status_6=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_6=rolling and wheel_status_2=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_6>0) and not ((mechanical_pedal_pos_R and
   (((wheel_status_3=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_3=rolling and wheel_status_7=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_3>0) and not ((mechanical_pedal_pos_R and
   (((wheel_status_4=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_4=rolling and wheel_status_8=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_4>0) and not ((mechanical_pedal_pos_R and
   (((wheel_status_7=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_7=rolling and wheel_status_3=rolling) or
   ground_speed=0 or not (control_system_validity)) and
   green_pressure_in_selector_valve=0))) implies
   wheel_braking_force_7>0) and not ((mechanical_pedal_pos_R and
   (((wheel_status_8=rolling or ground_speed=0) and
   green_pressure_in_selector_valve>0) or
   (((wheel_status_8=rolling and wheel_status_4=rolling) or
   ground_speed=0 or not (control_system_validity)) and
```

```
    green_pressure_in_selector_valve=0)))                       implies
    wheel_braking_force_8>0)));
```

The same assume/guarantee structure is used, and this is further refined by invoking the specifications of the contained blocks using the name matching of contract names. This refinement is repeated iteratively all the way down the hierarchy to the leaf components so that all specifications are shown to be satisfied. If they are not, then the NuXMV model checker produces a counter-example identifying the conditions that cause proof failure. This information can then be used to identify the cause of the failure and develop a design correction. The corresponding refinement is shown below. Note the name matching. This specification invokes contained component formal requirements specifications through the component dot contract syntax.

```
CONTRACT never_loss_of_all_wheel_braking
--subcontract
   REFINEDBY
      phys_sys.never_loss_of_all_wheel_braking,
       phys_sys.getting_green_pressure_in_selector_valve,
       ctrl_sys.system_validity,
       ctrl_sys.expected_behavior_as_cmd_pair_4_8,
       ctrl_sys.expected_behavior_brake_as_cmd_8,
       sensor_8.sensor_translation,
       ctrl_sys.expected_behavior_brake_as_cmd_4,
       sensor_4.sensor_translation,
       ctrl_sys.expected_behavior_as_cmd_pair_3_7,
       ctrl_sys.expected_behavior_brake_as_cmd_7,
       sensor_7.sensor_translation,
       ctrl_sys.expected_behavior_brake_as_cmd_3,
       sensor_3.sensor_translation,
       ctrl_sys.expected_behavior_as_cmd_pair_2_6,
       ctrl_sys.expected_behavior_brake_as_cmd_6,
       sensor_6.sensor_translation,
       ctrl_sys.expected_behavior_brake_as_cmd_2,
       sensor_2.sensor_translation,
       ctrl_sys.expected_behavior_as_cmd_pair_1_5,
       ctrl_sys.expected_behavior_brake_as_cmd_5,
       sensor_5.sensor_translation,
       ctrl_sys.expected_behavior_brake_as_cmd_1,
       sensor_1.sensor_translation,
       sensor_pedal_R.pedal_position_translation,
       sensor_pedal_L.pedal_position_translation;
```

In the above, phys_sys, ctrl_sys, sensor_1-8, and sensor_pedal_L/R are contained components of the top-level system.

The above assume/guarantee is a specification of what is assumed about the environment and what guarantee the block satisfies. It remains to specify the logical function of the block (i.e., its implementation) in a way recognizable to the NuXMV model checker. This is performed by constructing a file for each component containing both the assume-guarantee formal specification and an implementation behavior specification. OCRA provides a command to partially generate this file (an SMV text file) creating a template for each component. An OCRA command is provided that takes each component contained in the single OSS and extracts all the required information except the implementation specification into a set of SMV files, one per component.

The implementation part of these files must currently be completed manually. Consider the following example for the “AdditionGate” component. The intended function of this component is to sum two inputs (integer values ranging from 0..10 [in arbitrary units of measure]) and output the lesser of the sum or a maximum of 10. There are no assumptions about the environment, and the guarantee says that the block guarantees that output = sum of inputs if the sum is < 10; otherwise it is 10. Note that this is not an implementation specification; it is a guarantee of performance. This distinction bears some thought because it is not at first intuitively obvious.

```
COMPONENT AdditionGate
   INTERFACE
      INPUT PORT in_1 : 0..10 ;
      INPUT PORT in_2 : 0..10 ;
      OUTPUT PORT out : 0..10 ;
   CONTRACT addition_behavior
   --the output of the addition gate is the result of the addition
   of the two inputs
   assume: true;
   guarantee: always (out = 10 iff ((in_1 + in_2) >= 10)) and always
   (out = (in_1 + in_2) iff ((in_1 + in_2) <= 10));
```

To complete the specification, the implementation must now be specified. This is done in an SMV file. The completed SMV file is shown below. The bulk of this was created automatically by OCRA excepting the ASSIGN block. This block represents the functional behavior of the implemented block; that is, if sum (in_1:in_2) < 10, then the output is sum (in_1:in_2); otherwise it is 10.

```
==============================================================
MODULE main
 VAR
 AdditionGate_inst : AdditionGate(in_1, in_2);
 VAR
 in_1 : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
 in_2 : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
 DEFINE
 out := AdditionGate_inst.out;
==============================================================
 End of module
==============================================================

==============================================================
MODULE AdditionGate(in_1, in_2)
 VAR
 out : {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
 LTLSPEC NAME addition_behavior_norm_guarantee := (TRUE -> ( G (out
   = 10 <-> in_1 + in_2 >= 10) & G (out = in_1 + in_2 <-> in_1 +
   in_2 <= 10)));

 ASSIGN
 out := (in_1 + in_2 < 10 ? in_1 + in_2 : 10) ;
================================================================
 End of module
```

The end result of all these steps is a system OSS file and a set of completed component SMV files. These enable us to begin the safety-analysis steps described next.

6.3 FTA

The end result of the system capture in SysML, the formal specification of requirements in logical formulas embedded in SysML profiles, and the automated translation processes is an OSS format text file. This is the file format ready to be processed by the OCRA tools. The analysis comes in two varieties: 1) contract-based safety analysis (CBSA) relies only on the formal specifications, and 2) MBSA extends the component models with failure extension annotations.

CBSA and MBSA are fundamentally fault tree reliability analyses, not safety analysis methods, because a safety engineer would understand the term and tangentially relate to safety. This confusion is prevalent throughout the literature; however, to avoid yet further confusion in terminology, the terms are used in the literature even though they are not strictly correct.

6.4 CONTRACT-BASED SAFETY ANALYSIS (CBSA)

At this stage, it is shown how OCRA takes the OSS file, performs a formal CBSA, and constructs a fault tree. Note that a CBSA fault tree does not contain probability information because it is embedded in the fault extension files that are not used by CBSA.

The OCRA toolset provides a number of commands to:

- Check the syntax of the OSS file for errors.

- Check consistency of the specifications.
- Check refinement of the specifications.
- Generate NuXMV templates.
- Generate proof traces and counterexamples in which specification proofs are violated.
- Generate CBSA fault trees from a violation of formal assume/guarantee specifications.

6.5 MODEL-BASED SAFETY ANALYSIS

At this time, completed templates (SMV) and a fault extension file (FEI) developed by FBK to demonstrate MBSA are being used.

For the moment, the fault extension files provided by FBK to create FTs are being used for the hazards identified in AIR-6110. The eventual goal is to capture the component fault information in the SysML diagram and translate it automatically to create the required text files. The fault information comprises a fault mode, an associated probability, and a type classification, such as permanent or transient.

Fault-extension information required for MBSA is provided to the NuXMV and xSAP tools in a different format to the OSS format. Creating those fault extensions is currently a partly manual process of translating the formal specifications into the xSAP language.

When the design has been debugged and the checks are successful, a NuXMV template file is automatically created for each leaf component. The templates provide most of the information but currently must be completed manually by adding expressions (logical formulas) for the expected implementation of the specifications. In addition, to produce full FTs, an additional FEI must be manually written incorporating the component fault information.

6.6 CASE STUDY CONCLUSIONS

The overall purpose of the worked example is to show the feasibility of developing formal requirements for components (i.e., hierarchical sub-systems of arbitrary complexity) using a formal language that supports composability. One of the concerns often expressed about formal methods is scalability. Its application to a wheel brake system has been shown, but this is a relatively simple aircraft function. It serves only to illustrate the principle involved. More work is needed to investigate the scalability issue.

To perform the initial FTA, at least a system architecture is required showing functional blocks. At this stage of system development, specifying any details about the actual implementation of the system is not necessary (i.e., whether the implementation is in hardware, software, or both). Only functional and safety requirements and component faults are discussed. When requirements are validated through the formal analysis, further decomposition of these requirements can be made into hardware and software requirements for the implementation teams. At this level, a more detailed FTA of hardware reflecting actual implementation is required to satisfy the current certification guidance. It is not believed that software failure or human error can or should be included in an FTA because there can be no justification for any probability that might be assigned.

Verification is accomplished in the usual manner against these requirements (e.g., DO-178C and DO-254).

The example shows how, provided the components satisfy their formally specified requirements, the system composition can also be guaranteed to meet its formal specification. In that way, a large system can be decomposed to smaller sub-systems (components), each developed and verified independently and then composed by the system integrator with high confidence that there are no missing, incomplete or contradictory requirements. Furthermore, the modeling environment allows the system integrator to assure that system safety requirements are met, DAL assignments are met, and the overall system is consonant with the applicable regulatory requirements.

7. TASK 2 TECHNICAL STANDARD ORDER AUTHORIZATION (TSOA)

7.1 TSOA PROCESS DEFICIENCIES

In many ways, the Technical Standard Order Authorization (TSOA) process is the antithesis of the considerations motivating this work. This report addresses integration concerns, whereas the TSO mechanism seeks explicitly to divorce the regulatory approval of a component's design and manufacturing from issues of its installation and integration aboard any given aircraft.

Nevertheless, devices approved under TSOAs, in particular complex and software-intensive electronic devices approved under TSOAs, have been widely included in system architectures that tend toward higher levels of integration with each succeeding generation of aircraft. This trend raises policy-level questions about continued reliance on TSOAs as certification data.

The original purpose of TSOs was to identify and standardize parts, components, and devices that could be moved readily from one aircraft installation to another, without redesign and with self-evident acceptability. This philosophy worked well and still works well for simple components whose operations and characteristics are straightforward and obvious.

A seatbelt, for example, works as well in a commuter turboprop as in a commercial jet transport. The seatbelt's TSO can fully specify all relevant expectations for the isolated assembly in question. Each new installation can, with trivial effort, confirm the applicability and suitability of the same TSO'd seatbelt in another aircraft type.

Similarly, a TSO'd aircraft tire relies on mounting geometry and loading, but it does not rely on the aircraft type. A TSO'd microphone for VHF radios relies on plug standards and impedance matching but it does not rely on the aircraft type.

In short, a proper TSO should be associated with a component or device that is recognizably independent of its installation/integration variables. In reality, however, what was witnessed over approximately the last 30 years is a gradual migration of TSO'd equipment into ever more highly integrated aircraft systems. This shift has, at least in part, been driven by a flawed understanding of the nature and purpose of TSOAs.

Some airframers and some avionics suppliers have focused almost obsessively on the notion that TSOAs are desirable always and everywhere; that if a product can be TSO'd, then it should be. A few industry players have even made a subsequent mental leap, insisting that if a gadget can be

TSO’d, then the FAA requires that all such gadgets aboard the aircraft be TSO’d. This is, of course, nonsense. A newly designed aircraft can in principle be approved and delivered with no TSO’d devices whatsoever.

The TSOA is merely one option for certain components. It is a choice. Like most choices, the decision to pursue a TSOA is subject to any number of tradeoffs, many of which are non-technical, such as retention of intellectual property or whether spares can be sold directly to operators.

Sadly, though, misunderstandings about the costs and benefits of TSOAs persist. Across much of industry, those misunderstandings might be getting worse.

The resulting difficulties have sometimes been aggravated by aggressive system designs that include some or all of the following: (a) subsets of capabilities described in one or more TSOs, (b) supersets of capabilities described in one or more TSOs, (c) functions not described by any TSO, and (d) combinations of multiple TSOs within one physical package or part number.

The FAA offers at least some administrative guidance to help sort through these possibilities. FAA Order 8150.1c “Technical Standard Order (TSO) Program” [53, Ch. 8] and FAA Advisory Circular 21-46 “Technical Standard Order Program” [54, Ch. 5] provide a basic framework to help applicants and FAA specialists make sense of essentially arbitrary combinations of functional allocations and equipment capabilities.

Although convenient in many ways, this guidance assumes that almost any such combination is valid and appropriate. For simple devices in a simple architecture, that assumption is surely defensible. However, at some point, the inherent complexity of merging a wide variety of disparate functions, even within a putatively tractable context—such as IMA—inevitably leads to loss of insight into system behavior, especially in the presence of random failures and design flaws.

TSO compliance adds nothing to that insight. TSOAs can produce complacency and a false sense of security among both manufacturers and regulators. If a product is TSO’d, it seems that less scrutiny can be rationalized. If someone elsewhere has approved the box, in-depth review may not be necessary.

Again, this is generally fine for simple devices and simple systems. As the complexity increases, however, this approach becomes increasingly questionable.

Rather than merely reacting to and automatically trying to accommodate the unrestricted ambitions of aircraft manufacturers, it is possible that the public interest might be better served by steering the industry’s understanding of TSO philosophy back to its roots.

The first-pass litmus test for any proposed TSOA should be a simple question: Could this newly TSO’d device be moved without modification to a different kind of aircraft? If the answer is yes, then the proposal very likely meets the spirit of the TSO mechanism. If the answer is no, then a closer examination of the customer’s intended usage of the TSOA is in order.

TSOAs imply nontrivial additional work for both manufacturer and regulator. If the intended benefit of getting a TSOA in the first place—portability across different aircraft—cannot be achieved by the specific TSOA(s) proposed, then that additional effort may be wasted. Although it is certainly possible to obtain a TSOA for a device that must be redesigned for each new installation, the value of such an approval is at best debatable.

The real threat, however, is tied to rising complexity. As system architectures grow more complicated, a proliferation of TSO’d components within those architectures is more likely to obscure emergent properties of the overall system. A retreat into point-designed uniqueness for all nonportable components is hardly a guarantee of design perfection, but doing so would at least avoid the distractions and superfluous considerations of forcing TSOA issues into situations that are not natural candidates for such choices.

Some FAA Aircraft Certification Offices (ACOs) have, in effect, fought a rear-guard action against misguided TSO applications by requiring an installation, a live and fully worked demonstration, under some vehicle-level approval, usually a supplemental type certificate (STC).

“Your TSO wishes are suspicious,” these ACOs seem to be saying to applicants, “and we will not grant your TSOA until you prove that everything is okay through an STC.” It is suggested that such an approach addresses the symptoms rather than the causes of the ACO’s original suspicions.

At a policy level, the most meaningful questions for a TSO applicant could and should be straightforward, and those questions can be based wholly on the reasons TSOs were conceived and drafted in the first place:

- Does the proposed device match a published TSO?
- Will the device comply with the TSO (or be granted appropriate deviations)?
- Can the device move to different installation environments without modification?

All these questions are addressed at some level in the guidance material above. In practice, however, there are differences in the treatment of TSO applications among different FAA offices, and odd combinations of subset/superset/non-TSO functions have been sanctioned in more than a few cases.

Admittedly, it is not the FAA’s job to design the system or the aircraft. But “can be TSO'd” is not synonymous with “should be TSO’d,” let alone “must be TSO’d.” This report contends that TSOAs represent a fragmentation of design assurance and certification responsibilities. When that fragmentation collides with architectural complexity, high levels of integration, and extensive sharing of critical resources, the odds of dangerous surprises go up.

Clarification and simplification of the nature and purpose of TSOAs would benefit both industry and the FAA. More rigorous screening of TSO-related proposals, with greater emphasis on the portability constraint, would help to steer many project plans away from choices that add no value and contain hidden vulnerabilities.

7.2 TSO-RELATED ACCIDENTS/INCIDENTS

The bulk of this report seeks to identify and characterize methods and tools that could improve the understanding of integrated airborne systems and the vulnerabilities that those systems might conceal. This subsection of the report seeks to connect those technological improvements with vehicle-level outcomes.

The connection between TSO’d equipment and known accidents or incidents is tenuous at best. For the purposes of this discussion, one should distinguish between older, more traditional federated-system architectures and newer, more highly integrated architectures. The latter are presumed to pose additional risks that might well be larger than is commonly supposed. That potential is explored in greater detail in section 7.1. By contrast, the discussion below assumes that the TSO’d components of interest exist within traditional federated systems.

Commercial transport aircraft have service lives that span decades, and the annual production rate of new aircraft is small compared with the size of the existing airline fleet. It is unsurprising that most accidents occur with older aircraft simply by virtue of their numerical dominance.

Therefore, most hull losses have been associated with older aircraft having traditional federated system architectures. That reasoning can be extended to say that the correlation between fleet representation and accident likelihood will persist.

The key assertion here, though, is that for traditional federated architectures, the TSOA status of equipment aboard accident aircraft is invariably unrelated to hazardous outcomes.

It is exceedingly difficult to find connections from any of the usual elements in accident chains—operator error, equipment failure, design error, maintenance problems—that lead to any TSO-related property of a traditional component or device. The problems do not lie within the TSO’d device, but elsewhere, always at a higher level of abstraction. An exemplar of this is the 2009 crash of Turkish Airlines Flight 1951 [6] concerning a B737-800 on approach to Amsterdam's Schipol Airport (see section 3.1.3). The accident report identifies the proximate cause as the failure of the left radar altimeter during final approach. This accident offers a classic example of the intellectual distance between devices and fatal accidents. The accident aircraft was delivered with a left/right pair of radio altimeters.
A published TSO exists for radio altimeters. The radio altimeters were critical elements in the accident scenario.

One question is if those radio altimeters TSO’d. Another is if their approval status played any part, however remotely, in the accident (i.e., if there was anything in the accident chain that might have been affected by a decision or an activity motivated or limited by the TSOA process).

Historically, Boeing avoided or took no certification credit for TSOAs. Even when TSOs that plainly covered a given component or device existed, the Boeing norm was to treat each part—usually an LRU of some sort—as unique to its intended aircraft. The developmental and production aspects of that part were handled under the umbrella of type certification (TC) and parts manufacturer approval (PMA).

Publicly available information in this area is limited. Perhaps newer Boeing models are more open to inclusion of TSO’d equipment. The Turkish Airlines accident aircraft referred to above is a relatively modern variant of the venerable B737. At least one type of LRU aboard that airplane—the radio altimeter—was eligible for TSOA and figured prominently in the accident chain.

There is no further information about the approval mechanism (TSOA, Letter of Design Approval, or PMA) actually applied to the pair of radio altimeters aboard the Turkish B737. However, one can usefully craft a worst-case scenario thought experiment as a lens through which to view the relevant parts of this accident.

Assume that both the left radio altimeter and the right radio altimeter aboard the accident aircraft had obtained TSOA under TSO-C87a, “Airborne Low-Range Radio Altimeter” [55]. This document references ED-30, a EuroCAE Minimum Operational Performance Specification (MOPS) [56] for an airborne low-range radio altimeter. MOPS typically detail performance requirements and a set of conformance tests.

Like most TSOs, TSO-C87a is a minimalist document. It specifies a set of basic requirements in equipment function and performance, in labeling of parts, and in data delivery, mostly by reference to industry standards from organizations such as RTCA, EUROCAE, and the SAE. Any device that meets these minimum requirements and can be replicated properly in production is eligible for the TSOA marking.

It is notable that there is no required software or hardware DAL specified by the TSO. The DAL requirement is stated to be dependent on the intended use in the end application and, therefore, it is up to the integrator to ensure that the DAL provided is sufficient for the intended application.

Details of the Turkish Airlines accident are covered in section 3.1.3 and [6]. For immediate purposes, the short version is that the flight’s problems began when its left radio altimeter failed during approach, indicating an incorrect altitude.

Any number of mitigations can be imagined that would cope sensibly and safely with such a failure. In the actual events, however, the Boeing-specified relationships among radio altimeter, autopilot, and autothrottle (i.e., the interactions of critical pieces of the system architecture governing the descent profile during a coupled approach) led to unexpected system behavior that confused the crew. Their resulting delays in comprehending and reacting to those surprises consumed the remaining margin for recovery. The aircraft was destroyed, nine people were killed, and scores were injured. It is questioned if this tragedy might have been prevented by improvements to one or more specific TSOs.

The short answer is no. It is implausible to imagine that anything in the TSO-C87a process could have interrupted Flight 1951’s deterioration of total energy on its final approach.

TSO-C87a, in effect, envisions the radio altimeter as a single-function LRU. Neither the FAA nor any of the standards referenced by the TSO address higher-level issues. The TSO has no knowledge of the number of radio altimeters aboard the aircraft. The TSO has no knowledge of the architectural arrangement of sensors, buses, integrity monitoring, or failure annunciation. The TSO has no knowledge of flight-manual procedures or emergency checklists.

The TSO has no knowledge of any such considerations. Those considerations must and should lie beyond the reach of the TSO. This is essentially a universal truth for accidents in which TSO’d equipment is involved. Whatever the problems were, they could not have been solved by better TSOs.

TSO-C87a must accommodate the full spectrum of radio-altimeter usage, from terrain advisory functions to Category-IIIb Autoland. Different operational requirements for TSO-C87a sensors will lead to different criticalities, different numbers of LRUs in any given shipset, and different architectural arrangements of system resources.

Those choices cannot be addressed within the TSO process. They must be addressed at the functional, system, and aircraft levels by the airframer and its suppliers. The TSO cannot and should not dictate the integrity levels of, for example, individual output monitors, or the nature of comparisons among redundant sensors. Those issues can be handled appropriately only at higher system levels and in the context of the airframer's cockpit philosophy.

Flight 1951's crash at Schiphol is a near-perfect idealization of the reasoning behind this thought experiment. The problems are not at the LRU level but at the system and the flight-ops levels. Any hypothetical improvement to a radio altimeter will be irrelevant if the autothrottle is willing to use a failed radio altimeter with neither meaningful annunciation nor useful mode transition.

This is the simple case. In reconstructing the Amsterdam accident, one has the luxury of simple devices, a simple architecture, and simple flight-deck procedures. The cockpit voice recorder and FDRs were easily and quickly retrieved from the crash site. The investigation was as straightforward as possible.

However, even in the simplest of cases, and even for popular aircraft—more than 9000 Boeing 737s of all variants have been built and sold—whose systems are exercised every day all over the world, flight crews still suffer the occasional tragic surprise.

In federated architectures, such surprises cannot be fixed by better TSOs. In more complex architectures—in particular, architectures reliant on high levels of integration and resource sharing—inclusion of TSO'd equipment must not lead to an unwarranted relaxation of scrutiny. Despite all the official disclaimers about TSOs having nothing to do with installation requirements, there are those in both industry and in various civil air authorities who approach TSO'd devices more casually than they would equivalent devices built from a clean sheet of paper and approved under TC/PMA. Although no one admits to such behavior, it is real and is a mistake—one with potentially catastrophic consequences.

Whether federated or integrated, the design limitations and goals remain the same:

- You can do only so much with any given LRU.
- All operational behavior must be simple and obvious to the flight crew.
- Systems must degrade predictably toward even greater simplicity and situational awareness, sacrificing performance if necessary.
- If increasing system complexity obscures anything in the three items above, then the system will hide its worst secrets from flight crews until the worst possible moment in service.

One should be able to follow the lines of reasoning established by improved methods and tools not just to better LRUs—TSO'd or otherwise—but all the way to flight-deck effects.

One way to organize that reasoning might be to organize it into three stages. First, examine each accident described in section 7 of this report through the prism of the four items listed previously, looking for violations in one or more of the four. Second, imagine how the proposed methods and tools discussed earlier in this report would have dealt with the links in the causal chains of those accidents. Third, map a judgment of the net effects of improved tools on those accidents to items 2 and 3 in the list above.

Such a view might lead to contemplation of alternative strategies, including a more draconian insistence on operational simplicity for many aircraft functions, systems, and operational procedures.

8. INTEGRATING LESSONS LEARNED

As discussed in the introduction, following the retirement of the former principal investigator, the latter stages of the research program were refocused to better integrate and represent the capabilities of the larger research team. Although significant value may be gained from the formal model-based safety analyses detailed in previous sections, a larger benefit may be gained by establishing how such techniques can be blended and integrated within the larger System and Safety Engineering processes. In addition, some lessons learned were derived from reviewing aviation accidents and incidents, and from some observed failures in other safety-critical systems. These lessons learned are described as follows, in no particular order. As a general observation, confirmation is found for the following statement made by NASA’s C. Michael Holloway, after his studies of aviation accidents:

> “To a first approximation, we can say that accidents are almost always the result of incorrect estimates of the likelihood of one or more things.”

In other words, most accidents in which system design was a contributing factor involved a scenario that the system designers thought could not happen. This was because they erroneously believed the probability was much less than the required upper limit on failure probability or they did not think about that scenario at all (essentially making the assumed probability zero). One mental foible that leads to erroneously low estimates of probability is the assumption that if a scenario is too complex to be understood, it must need a very rare set of conditions for the scenario to occur.

One common problem was the misinterpretation of an inter-subsystem signal, either the full definition of the signal’s meaning and/or its quality. For example, does a signal that says that TRs are retracted include that they are locked. An example of lacking quality notification is the Turkish Airlines B737 that crashed near Amsterdam in 2009.

A related problem is the lack of understanding of what fault effects can befall a signal. Two of the most common fault effects that are insufficiently considered are intermittent faults (of specific characteristics, such as pulse width, repetition rate) and Byzantine failures. An example of the former is the Qantas Flight 72 incident, and examples of the latter include the Leisure Airways A320 brake and space shuttle incidents. An important observation is that all the most common fault-tolerance mechanisms (COM/MON and other output monitoring) are susceptible to Byzantine faults.

There are many accidents attributed to incorrect crew action, for which system information given to the crew was insufficient or misleading. Examples include Air Inter Flight 148, American Airlines Flight 965, Air France 447, and Virgin Atlantic A340 flying HKG-LHR in 2005.

Some accidents were caused by a failure in one system coupling to another system that should have been unrelated. These unintended coupling mechanisms include fire, shorting of wires in a cable, and shrapnel. Examples include Swissair Flight 111, Sioux City DC-10, and Qantas A380. Shrapnel does not only consist of two pieces of metal flying off an engine. It could be the casing of a capacitor.

Some problems are well known, but still recur. The main example of this is pitot tube blockage (Air France flight 447, Austral Líneas Aéreas flight 2553, A-12 Oxcart reconnaissance plane, B-2 Spirit bomber, Northwest flight 6231, Aero Peru flight 603, Birgenair flight 301, F-16).

Many redundant systems use a hybrid NMR architecture in which a group of components can "vote out" a member of a redundant set. However, to do this correctly, the voters must use a Byzantine secure voting mechanism (which is rarely, if ever, done) and (because the goal of a hybrid NMR architecture is to tolerate more sequential faults than the simultaneous faults that can be tolerated by a simple NMR architecture) the "vote out" must be persistent until the failed component is repaired. An example of a lack of persistence is the Malaysian B777 incident near Perth in 2005. Another thing to note is that many system designers incorrectly use FTA for hybrid NMR designs when the needed tool is Markov modeling.

A well-known problem is the possibility of math errors, such as divide-by-zero, when the aircraft is positioned at some cardinal points, such as the north or south poles. Less well known is that these errors can occur at other aircraft positions with much higher probability, such as being directly over a waypoint. These math errors can cause exception handler executions, which need to be correctly accounted for in partitioned processors. If these executions are not handled correctly, partition violations can occur, such as a level-C partition causing loss of level-A functionality in a Bizjet PDU.

Accidents have been caused by energy or signals traversing a path (e.g., electrical wire or hydraulic pipe) opposite of that intended. Examples include TWA Flight 800 and a police van in which the flashing lights caused the transmission interlock to fail, killing two people and maiming eight.

There have been accidents and incidents in which components failed in ways that the system designers did not think were possible, but could be seen as highly probable by those with the proper expertise. Some of these failure modes include one electrical part turning another electrical part (e.g., a diode becoming a capacitor, capacitors becoming resistors, transistors becoming silicon-controlled rectifiers [SCRs], amplifiers becoming oscillators) and electrical parts created out of nothing ("parthogenesis"). The latter is usually due to a shift in parasitic electrical properties. In one case, an avionics cooling fan becoming an air-conditioner and the resulting condensation caused all the aircraft's flight control processors to short out.

Interrupts and other sources of temporal non-determinism are a major source of problems. Examples include a helicopter flight-control system in which floating point values were corrupted and a military quality processor that completely locked up because of a bounce in an interrupt line.

8.1 ADDITIONAL REFERENCES FOR SYSTEM FAILURE NASA VVFCS-DASHLINK

On a prior NASA-funded research contract [57], the Honeywell team created a set of system-failure descriptions for failure modes not considered by most avionics designers, but which have actually occurred. These descriptions are available as a slide set at c3.nasa.gov/dashlink/resources/624. The following commentary provides a reference of the material, which may be very pertinent to readers of this report. Although some of these issues have been mentioned previously in descriptions of accidents, incidents, or lessons learned, additional detail in these NASA web pages are included here. Also included are observations that proper design methodologies and tools could have prevented many of these problems.

Slide 3 provides definitions for a failure mode known as the Byzantine Generals Problem. Slides 4–10 describe a Byzantine failure that occurred aboard the Space Shuttle during the launch countdown for mission STS-124, which caused the launch to be canceled and incurred a cost of $200 million. These slides also describe a previous Byzantine failure aboard a Space Shuttle that should have alerted designers to its possible existence; the design had to be changed to accommodate these failures. Slide 11 shows a Byzantine failure in a mid-value select fault-masking design, which shows that this form of fault masking is susceptible to Byzantine failures. Slide 12 describes a Byzantine failure of a command-monitor (COM-MON) processor pair, which shows that all COM-MON architectures are susceptible to this failure. Slide 15 describes a number of Byzantine failure instances aboard a commercial aircraft that came close to grounding that fleet of aircraft, at a cost of nearly $10 million a day. All these Byzantine failures could have been prevented by correct architecture design. A proper design-analysis tool would have found the flaws in the architectures of these systems and alerted the designers to what corrective action would need to be taken.

Slide 13 describes a large number of Byzantine failures that occurred in a laboratory test of a network that was designed to be Byzantine fault tolerant. This testing showed not only that Byzantine faults can occur at a high rate, but also that Byzantine faults cannot be covered by in-line error-detection mechanisms, such as parity, checksums, or cyclic redundancy checks (CRCs). A proper design-analysis tool would have found the flaw in this design. It also would reject any fault-tolerant architectures that relied solely on CRCs or other in-line error checking for fault detection.

Slides 25 and 26 describe failure propagation due to the improper power supply distribution design. Again, a proper design-analysis tool could have found these fault-propagation paths and alerted the designers to corrective action.

Slide 28 describes a single point of failure that a proper design-analysis tool could have found.

Slides 31 and 32 describe failures due to architectures that made the unwarranted assumption that the only failures were "stuck-at" type failures. A proper design-analysis tool would not accept this erroneous assumption.

Slides 47–51 describe how an incorrect linked list structure caused a processor to stop operating. Slide 52 describes how an incorrect stack pointer value causes a processor to stop operating. These failures could be classified as "halt and catch fire" data structures. These failures show that "robust partitioning" and "time and space partitioning" cannot be made to work on all processors. A proper design-analysis tool would require designers to provide proof that a processor does not have any "halt and catch fire" data structures or instructions (such as the Intel F00F bug) to substantiate a claim of "robust partitioning" or "time and space partitioning." (Operating system vendor hype is insufficient rationale.)

Slide 58 gives a list of pitot failures. A proper design-analysis tool would ask a designer if the pitot tubes had any common mode failures (ice, tube blockage, failure to heat when needed).

8.2 ADDITIONAL OBSERVATIONS ON MBSA

As part of the second phase of research, an alternative model-based safety analysis of the AIR 6110 Wheel Brake system was conducted using AADL [58]. In this approach, the AADL Error Annex [20] was employed to make annotations to the system. In the previous public example, the error annex annotations used the composite error model provisions within AADL. These annotations largely comprised the hierarchical encoding of the structure of the fault tree within AADL component layers. Hazards were also annotated within the AADL model. In the original analysis [47], there were few provisions to check the declarations of the composite error mode with respect to the subcomponent structure or the lower-level error state machine annotations. This led to the potential for inconsistency between the annotations. For example, it may be possible in the composite-error model annotations to assume a parallel relationship among lower-level components, which are actually connected in series. In more recent work [59], a new error-flow-based analysis improves this situation.

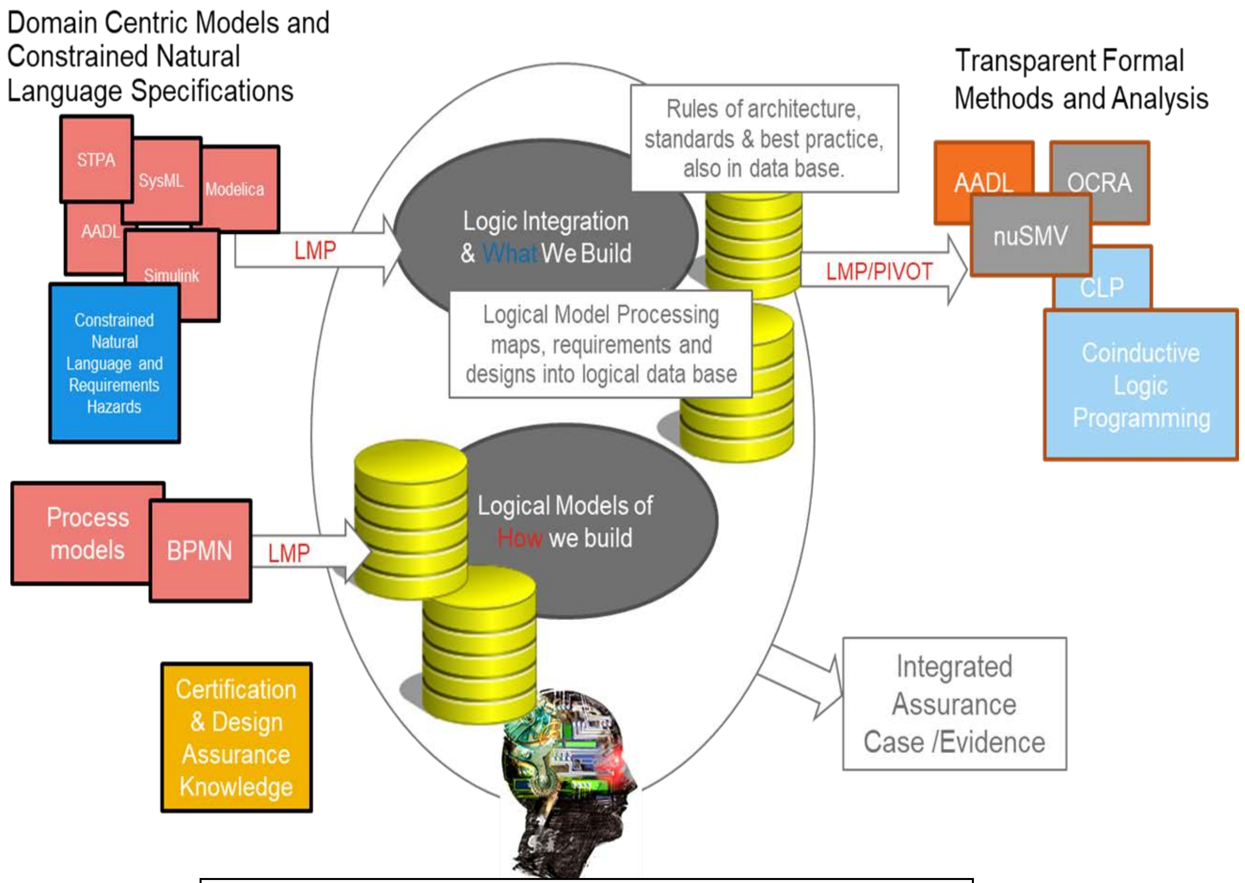
For both the OCRA and AADL approaches, it is interesting to note that the circuit aspects of the hydraulic configuration are largely omitted. In the AADL models, neither the hydraulic check valves nor the fuses are included. In the OCRA model, although the fuses are included, they are abstracted to a simple pipe, and their respective failure modes are limited to being stuck closed. Failures downstream of the fuse, such as brakes or leaks in the hydraulics, are abstracted. The behavior of check valves that prevent erroneous backward flow are also omitted. Considering the fundamental safety aspects of such hydraulic-related failure modes, it is interesting to note the omissions in both model-based safety analysis approaches.

Similarly, the treatment of the pedal validation logic also is brief in both existing models. Pedal sensor redundancy is minimal. Given the potential for single points of failure of such commands to result in inadvertent braking commands, the lack of redundancy with respect to both command availability AND integrity is surprising. This observation raised an additional concern with respect to the use of such automated analysis approaches. Given the large amount of data generated by these methods and the level of effort applied to review such data, simple oversights in the model abstraction may happen—the analysis is simply overwhelmed by data. If the problem (abstracted model) is not suitably formed, a large amount of effort may be expended, yet dominant failure scenarios may be missed.

In section 10.2, an alternative approach model-based safety analysis is briefly investigated, leveraging abstractions derived from circuit theory. In these models, alternative pedal-validation strategies are also investigated. It is interesting to note that the principled design and architectural refinement, which will be introduced in later sections, also addresses some of the architectural and redundancy strategies in a more systematic manner.

9. A VISION FOR COMPUTER-ASSISTED INTEGRATED SYSTEM AND SAFETY ENGINEERING

Over the past few years, the world has witnessed exponential growth in the application of Artificial Intelligence (AI) technology in almost every discipline. From medical to jet engine diagnosis, one is bombarded daily with the ever-increasing capacities of the IBM® Watson, Apple's SIRI, and related technologies. A relevant question is "Why are such technologies not applied to system and safety engineering?"—especially given the challenges identified in this research. One complication is the difference between big-data and small-data AI strategies. Big-data AI leverages statistical inference techniques that are well suited to large data applications, such as speech and image recognition. However, as argued by Haley [60], such approaches do not lead to a true understanding of the concepts and relationships of the domain. Instead, more fundamental logical reasoning is required. An interesting experiment toward establishing a system that supports such logical reasoning is the AURA project [60] and the related "Inquire Biology" textbook [61]. These projects show the advantages of encoding information in a logical database, enabling AI support engines to greatly assist the learning processes. This AI support allows readers to ask natural language questions from the book and allows the book to generate questions for the readers. The proof of concept demonstrations also show the technology's ability to accommodate complex subjects, such as biology and physics. Relative to the previously identified challenges and with respect to the human ability to be able to effectively reason about the increasing levels of complexity, one may postulate how such technologies may be used to support humans. Leveraging the structured notations from model-based system engineering, such as SysML, AADL, and STPA , it may also be possible to generate a large part of the logical model automatically using logical model processing (LMP) approaches. Figure 12, an integrated vision for machine assisted integrated system and safety engineering, shows the potential for such an approach.


Domain Centric Models and Constrained Natural Language Specifications
STPA
SysML
Modelica
AADL
Simulink
Constrained Natural Language and Requirements Hazards
LMP
Logic Integration & What We Build
Rules of architecture, standards & best practice, also in data base.
LMP/PIVOT
Transparent Formal Methods and Analysis
AADL
OCRA
nuSMV
CLP
Coinductive Logic Programming
Logical Model Processing maps, requirements and designs into logical data base
Process models
BPMN
LMP
Logical Models of How we build
Integrated Assurance Case /Evidence
Certification & Design Assurance Knowledge

Using integrated Logical Models, can we insert logical reasoning and Artificial Intelligence to remove known issues and identify deviations from accepted practice to support integrated reasoning of both **what** and **how** we build?

Figure 12. An integrated vision for machine-assisted integrated system and safety engineering

In parallel with this research effort, Honeywell has also developed and refined a constrained natural language-based approach to requirement and hazard specification by extending and enhancing the industrially pragmatic EARS approach [62]. By formalizing the key aspects of the requirements approach, the transparent insertion of formal method technologies, such as model checker and SMT solvers, toward requirements consistency and completeness analysis, and automated test generation, has been achieved. Integrating this technology with SysML/STPA models may also yield significant benefits. Similarly, extending the constrained natural language requirements approach to support the natural language specification of the OCRA and SysML models, as explored during the first portion of this research, may also improve the industrial traction of such formal methods technologies. The logical model foundation may also be used as a "transparent pivot" to support the insertion of other formal method technologies, such as model checkers and coinductive logical analysis. Given the growing complexity of current and emerging systems, it is believed that both formal methods and automated assisted reasoning are becoming increasingly necessary. Interactive, guided logical assistance and model-based safety analysis may also support the human as he/she performs safety analysis and assurance tasks, such as the STPA causality analysis.

In addition to the automation and assistance to support "what we build," it is also believed that the same technology can be applied to "how we build." In modern, globalized workplaces, the complexity of the organizational structures developing the next generation of systems is growing increasingly more complex than the products themselves. Therefore, it is proposed that the logical model framework should be extended to incorporate formal logical modeling of the development processes and design assurance objectives. For this, it is envisioned that ARP- and RTCA-derived process steps could be formally captured using standard notations, such as Business Process Model Notation. Prior research [63] has already demonstrated the conversion and benefits of converting such models into a formal logical form. It is believed that integrating such models into a common logical framework, which also includes the safety and architectural refinement models of the previous section, will yield a unique end-to-end capability. It will support the integrated logical reasoning over safety, implementation, and process models. By incorporating additional rules into the support infrastructure, it also will be possible to capture lessons learned and standardized best practices and guidelines.

Incorporating constrained natural language capabilities to support the query of the integrated the logical models, it is envisioned that (in a final form) the system will be available to support both model (product) and development process audits. For example, a typical query may be "how is this hazard mitigated?" The system response will then yield an integrated response linking all the STPA-derived safety constraints with the architectural failure mitigations, the associated DAL specific process objectives, and assurance evidence. This response would be in the form of reviewed test and analysis artifacts. Given the growing complexity of systems and the increasing need for more flexible assurance reasoning to address the new technologies, such as machine learning and increasingly autonomous systems, the integrated logical framework of figure 12 may be beneficial.

As discussed in section 8.1.2, one of the risks associated with model-based safety analysis and the application of formal methods techniques (as explored during the first portion of this research) is the fidelity of the model with respect to the physical real world. In many of the proposed techniques, the failure propagation models are abstracted to a degree that the physics of the underlying system are lost. Therefore, integrating concepts of physics into the integrated logical database may also present unique value. The standardization of physical modeling language profiles, such as Modelica with SysML [36], may also be used to mitigate some of these risks because they allow the physics to be introduced elegantly into the system-engineering environment.

Given the standards and technologies that exist today, it should be possible to link the assumed failure-mode assumptions of the safety models with parametrized equivalence-class-based fault injection or "physical mutations" within the physical models. Additional rules, such as circuit conservation, can additionally be used to support system causality and failure-propagation analysis. Such an integrated model may then be leveraged toward safety-aware, case-based, reasoned physical test generation in areas of perceived vulnerability or risk. Instrumenting the physical design models, system stress margins, and targeted coverage metrics may then be derived.

This would be done under the integrated guidance of the expert system, which maps the resulting evidence to the underlying assurance arguments.

Some investigations and experiments targeted to explore the feasibility of such a system are presented in section 10.

10. EXPLORING ENABLING TECHNOLOGIES

10.1 LOGICAL MODEL PROCESSING OVERVIEW

Before describing the details of the work, the (LMP) [64, 65] approach that provides the foundation for the logical integration of heterogeneous models is briefly outlined. LMP is an approach that applies logic programming to model-based system development. It is an approach that converts models and software into a logical predicate form, which then supports logical query. This allows very simple declarative queries to be developed that can interact with and query the model. LMP is based on the use of the Prolog language to formally specify rules to be applied to an appropriate representation of the applicative model. This representation of the model is composed of Prolog facts. Prolog is a declarative language that is used to express rules applied on predicates. Rules can then be combined using Boolean Logic. Prolog syntax is very simple, and most programs can be specified using AND, OR, and NOT logical operators.

LMP consists of a methodology and a set of tools and Prolog libraries.

The LMP methodology includes the following principles to support model-driven engineering:

- The meta-model classes define Prolog fact specifications, whose parameter's names correspond to the attributes and names of the classes.
- An instantiated model resides in a populated Prolog facts base in which facts, parameters, and values correspond to the classes' attributes and values.
- The model-processing program is expressed as a set of Prolog rules whose predicates are other rules or facts.
- Executing an LMP program simply requires the facts base associated with the model instance to be merged with the rules base associated with the processing to be performed, and subjecting the composite model-related queries to the Prolog interpreter.

With the Prolog language being an ISO standard, any Prolog environment can be used to support the LMP approach. The one that has been used in the commercial LMP tool chains is SB-Prolog. In this investigation, SWI-Prolog was also used. This was done partly to test the ease at which different Prolog technologies could be intermixed and partly to take advantage of the more modern IDEs available for SWI-Prolog.

In addition to the Prolog interpreter itself and its run-time environment, a set of additional tools and reusable libraries are also part of the LMP toolbox. These include, in particular, input model parsers and output model unparsers (or printers). The currently available parsers are for XML/XMI models (xmlrev), AADL models (aadlrev), and programming languages. Facts-based generation also can be implemented for memory-stored models handled by modeling tools.

The main benefits brought by the LMP approach are:

- A clear separation between the model to be processed (facts base) and the model-processing program (rules base).
- A strong traceability between model-processing requirements and their implementation (one rule per requirement).
- The declarative and logical programming style offered by the Prolog language.
- The ability to define a modular set of processing rules and to link them together at run time.
- The ability to use the same implementation language for all kinds of model processing (i.e., navigation within the model language constructs [query language], verification of model properties [constraint language], and transformations [transformation language] for model-to-model, model-to-text, and text-to-model).

A key attraction of the approach is the ability to apply rules in the form of lessons learned or best practice to the models under development. For example, where known issues exist, such as the inadequacy of inline coverage codes (e.g., CRC) relative to the assumed communication channel fault models, simple rules and corresponding queries can be developed to detect such issues at design time. Similarly, more elaborate queries, such as an Architectural Byzantine Vulnerability Analysis, can also be developed by integrating multi-model perspectives (e.g., structural, logical, and safety perspectives) into the integrated logical model.

The last point is of particular interest. Working in an ISO standard language to implement core model processing technology maximizes reuse across different tool-chains while avoiding lock-in/duplication, which accommodate different vendor API and their versions. In this work, LMP-based technology was applied in two areas. First, LMP was applied to the OCRA-based wheel brake system case study. Here, the technology to generate the OCRA model from the annotated SysML Model was used. Second, the feasibility of translating an STPA model (captured in the SAHRA [67] tool) to predicate form was investigated and confirmed. SAHRA, a commercial tool, was targeted to support STPA modeling. It was developed by the Applied University of Zurich. SAHA is an extension to the Enterprise Architect [68] (SysML/UML) tool chain. One of the attractions of this tool is that it enables the STPA models to be extended and related to requirements and architectural refinement (logical and physical) models captured using SysML. The original intention was to link this model to the work presented in the next three sections to show the benefits of STPA/SysML model integration. However, this was not possible within the constraints of the remaining research budget.

It should be noted that a longer-term goal is to also apply a similar logical transformation to models of the development processes (e.g., translating 4754A-based development and design assurance process models encoded in Business Process Modeling Notation) [69]. Hopefully, the logical representation of both the process and model artifacts will support the integrated reasoning and querying of both product and process artifacts. It is believed that the integrated representation will support a firm foundation for design assurance argumentation. Although much of this vision is beyond the scope of this effort, the above vision is shared to establish the context for some of the work that follows, specifically why the applicability of the logical model processing technology to the STPA is being explored.

For more details of LMP, refer to [65, 66]. Similar approaches have also been developed by Storrle [70], and Almiendros-Jimenez [71].

10.2 EXPLORING CIRCUIT AWARE MODEL-BASED SAFETY ANALYSIS WITH CONSTRAINT LOGIC PROGRAMING

This section will present the fault-tree generation using a Constraint Logic Programming (CLP) approach. This outlines the experiments to explore the potential for physical-model reasoning within the logical-modeling framework.

10.2.1 Motivation

In recent years, many new technologies to support MBSA have been emerging and gaining traction within the industry. The next revision of 4761 (4761A) currently under draft is also provisioning for the use of MBSA techniques within the certification framework. In section 8.1.2, observations with respect to two current approaches towards MBSA are discussed.

A major observation related to the prior studies highlights the fact that some of the key components with respect to hydraulics fault-isolation and mitigation were missing. These were check valves and hydraulic fuses. These omissions are concerning, given that such components form the foundation of the hydraulic circuit fault-containment. It may be argued that such omissions were simply due to the content of the original AIR [50], which also abstracted such components. However, with the level of effort devoted to the safety analysis, it is concerning that the need for such components did not surface during the analyses. Additionally, given the significantly larger amount of data generated by the formal analyses, a further concern is that such omissions may be easily overlooked as safety engineers get swamped and overwhelmed with large automatically produced data sets.

To address these concerns in this appendix, an alternative approach to model-based safety analysis was explored, which is based on a finer abstraction to the underlying physics and circuit-based behavior of the system.[1]

10.2.2 An Overview of Constraint Logic Programming Approach to MBSA

The techniques presented in this section were first developed under previous research funded under the NASA Aviation safety branch, as part of the Certware [42] project that was executed in conjunction with Eric Bush of Kestrel Technology [72]. The basic ideas behind the approach is to adopt a physics-aware approach to model-based safety assessment to mitigate some of the issues identified above. The new method applies a Prolog-based CLP approach toward model-based safety analysis. CLP(*) systems extend the normal unification operation of normal logic programs and substitute a specialized unification operation when the variables are of their parameterized type. The asterisk is replaced by a type indicator. CLP(*R*) provides a constraint solver over real-valued variables, carrying collections of equality and inequality constraints over each variable that

[1] It is our view that physic laws form the basis of fault propagation, and therefore provide a foundation for fault–propagation deduction.

has (so far) satisfiable constraints. This allows for modelling the basic physics of continuous systems. Basic circuit laws such as Ohm’s law are also encoded into the component definitions.

```
 resistor(electrical(V1,I1),electrical(V2,I2),R,ok):-
   {I1= -I2,
    V1-V2=I1*R}.
```

The last parameter of the resistor predicate is used to bind resistor failure state. If the resistor is *ok*, the constraints are expressed in terms of the parameterized resistance. However, alternative predicates are added for the failure modes of the resistor, such as *open_circuit* or *short_circuit*, as shown below. These annotations correspond to the “mutations” with respect to the normal component constraints

```
 resistor(electrical(V1,I1),electrical(V2,I2),_,failed(open_circu
   it)):-
   {I1= -I2,
    V1-V2=I1*R2,
    R2 =100000000.0}.


resistor(electrical(V1,I1),electrical(V2,I2),_,failed(short_circu
   it)):-
   {I1= -I2,
    V1-V2=I1*R2,
    R2 = 0.0001}.
```

Similar annotations can be added for other components, such as batteries, switches, and lamps for the electrical domain.

```
battery(electrical(V1,I1),electrical(V2,I2),Voltage,ok):-
   {I1+I2=0, Voltage=V1-V2,
   V1 >= V2
   }.

battery(electrical(V1,0),electrical(V2,0),_Voltage,failed(dischar
   ged)):-
   {V1-V2 = 0}.


switch(electrical(V1,I1),signal(0),electrical(V2,I2),ok) :-
 {
 I1 + I2 = 0,
 V1-V2=I1*100000000.0
 }.

switch(electrical(V1,I1),signal(1),electrical(V2,I2),ok) :-
 {
 I1 + I2 = 0,
 V1-V2=I1*0.0001
 }.

switch(electrical(V1,I1),signal(_),electrical(V2,I2),failed(short
   )) :-
 {
 I1 + I2 = 0,
 V1-V2=I1*0.0001
 }.

switch(electrical(V1,I1),signal(_),electrical(V2,I2),failed(open)
   ) :-
 {
 I1 + I2 = 0,
 V1-V2=I1*100000000.0
 }.
```

The components are then instantiated in accordance with the circuit topology. For the initial illustration, an example from the draft of ARP 4761A is shown in figure 13, followed by its predicate representation. The circuit consists of a 10v battery, two switches, a lamp bulb, and a ground connection.

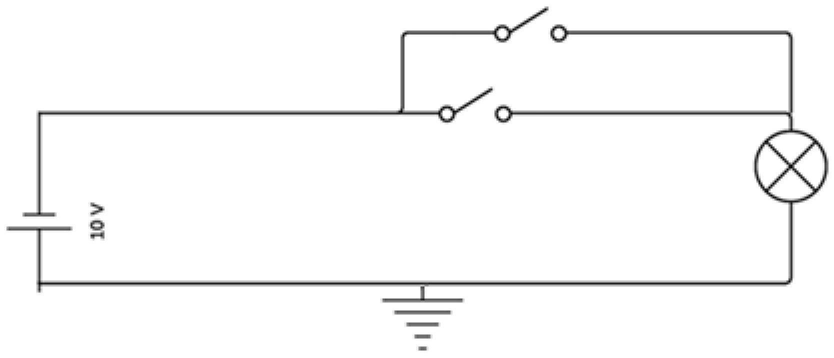


Figure 13. Simple 4761A example circuit

```
analog_circuit_4761A_tree((Sw1State,Sw2State,LampState,BatState),
  Obs):-
component(1, switch(VC1,signal(1),VC3,_), Sw1State),
component(2, switch(VC2,signal(1),VC4,_), Sw2State),
component(the, lamp(VC5,VC6,Obs,_), LampState),
component(the, battery(VC0,VC7,12,_), BatState),
connected([VC0,VC1,VC2]),
connected([VC3,VC4,VC5]),
connected([VC7,VC6]).
```

The prolog CLP(R) environment can then be used to extract all faults that can lead to a “dark state” with a simple query, as follows:

```
?- min_fault_tree(analog_circuit_4761A_tree(U,dark),U,T).

T  =  or(and(fault('switch  1  failed(open)'),  fault('switch  2
   failed(open)')),   fault('the   battery   failed(discharged)'),
   fault('the lamp failed(open_circuit)'))
```

The corresponding fault tree is shown in figure 14:

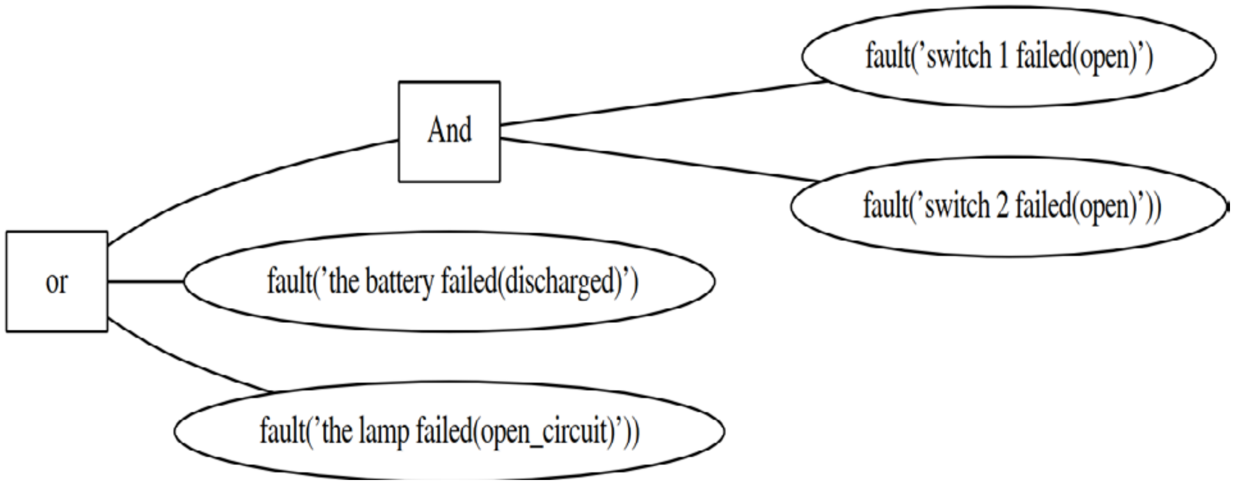

or
And
fault('switch 1 failed(open)')
fault('switch 2 failed(open)'))
fault('the battery failed(discharged)')
fault('the lamp failed(open_circuit)'))

Figure 14. CLP deduced fault tree for omission (dark)

It should be noted that no additional fault-propagation annotations need to be added to the component models. In addition, the annotations within the CLP models are component centric, and they do not constitute the structure of the fault-tree. Instead, the structure is derived from circuit topology and encoded component constraints. To show this further, an additional switch can be placed in series with the parallel switches, as shown in figure 15.

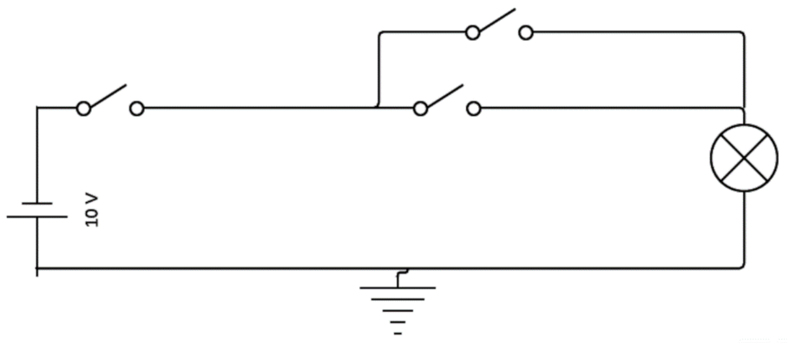


Figure 15. More complex circuit configuration

The corresponding predicate for the circuit is shown below:

```
 analog_circuit_4761A_tree((Sw1State,Sw2State,LampState,BatState)
   ,Obs):-
component(1, switch(VC1,signal(1),VC3,_), Sw1State),
component(2, switch(VC2,signal(1),VC4,_), Sw2State),
component(the, lamp(VC5,VC6,Obs,_), LampState),
component(the, battery(VC0,VC7,12,_), BatState),
connected([VC0,VC1,VC2]),
connected([VC3,VC4,VC5]),
connected([VC7,VC6]).
```

As above, a simple query can be used to explore the potential contributions to the failure state.

```
min_fault_tree(analog_circuit_II_tree(U,dark),U,T).

T  =  or(fault('a_battery  5  failed(discharged)'),  fault('a_lamp  4
   failed(blown)'),    and(fault('a_switch    1    failed(open)'),
   fault('a_switch    2    failed(open)')),    fault('a_switch    3
   failed(open)'))
```

The corresponding fault tree is shown below in figure 16:

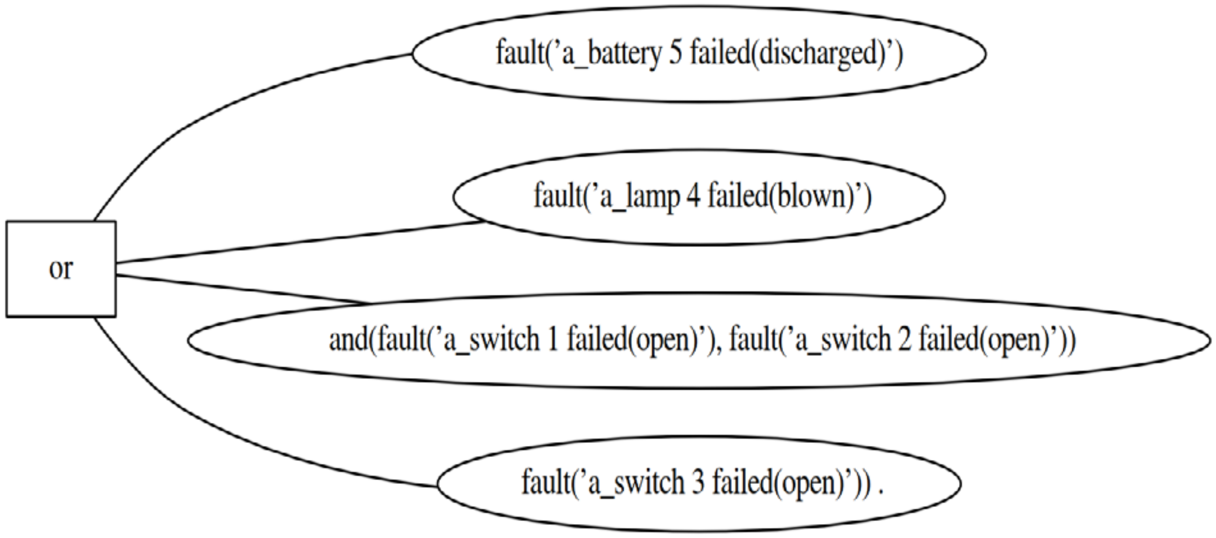

or
fault('a_battery 5 failed(discharged)')
fault('a_lamp 4 failed(blown)')
and(fault('a_switch 1 failed(open)'), fault('a_switch 2 failed(open)'))
fault('a_switch 3 failed(open)')) .

Figure 16. CLP deduced fault tree for omission (dark) with additional switch 4 for blown lamp

Two additional circuit configurations that were used to explore this model-based safety analysis are shown in figure 17. These circuits where included to explore the difference between omission and commission failure modes. This was done to show how such tools may help engineers deal with the complexities that often arise as functional availability and integrity are traded off. For illustrative purposes, consider the lamp symbol ⊗ to be an actuator that is safety critical. Is it better to include the additional connection between the switches of the second configuration or not? Exploring the circuit configurations for both commission (on) and omission (off) hazards can provide valuable insights.

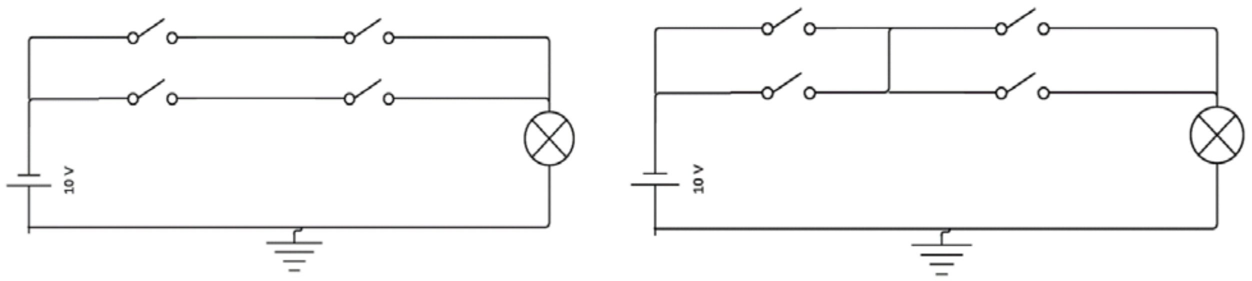


Figure 17. Example circuits III and IV

The circuits are realized using similar circuit predicates. However, in this instance, to explore both commission and omission-failure scenarios, additional command parameters (shown in red) are needed to denote the nominal positions of switches.

```
analog_circuit_III_tree((Sw1State,Sw2State,Sw3State,Sw4State,Lamp
   State,BatState),SW1,SW2,SW3,SW4,Obs):-
 component(1,switch(VC1,signal(SW1),VC3,_),Sw1State),
 component(2,switch(VC2,signal(SW2),VC4,_),Sw2State),
 component(3,switch(VC5,signal(SW3),VC6,_),Sw3State),
 component(4,switch(VC7,signal(SW4),VC8,_),Sw4State),
 component(5,lamp(VC9,VC10,Obs,_),LampState),
 component(6,battery(VC11,VC12,12,_),BatState),
 GND = electrical(0,_),
 connected([VC8,VC4,VC9]),
 connected([VC2,VC3]),
 connected([VC7,VC6]),
 connected([VC11,VC1,VC5]),
 connected([VC9,VC4,VC8]),
 connected([VC10,VC12,GND])
 analog_circuit_IV_tree((Sw1State,Sw2State,Sw3State,Sw4State,Lamp
   State,BatState),SW1,SW2,SW3,SW4,Obs):-
 component(1,switch(VC1,signal(SW1),VC3,_),Sw1State),
 component(2,switch(VC2,signal(SW2),VC4,_),Sw2State),
 component(3,switch(VC5,signal(SW3),VC6,_),Sw3State),
 component(4,switch(VC7,signal(SW4),VC8,_),Sw4State),
 component(5,lamp(VC9,VC10,Obs,_),LampState),
 component(6,battery(VC11,VC12,12,_),BatState),
 GND = electrical(0,_),
 connected([VC8,VC4,VC9]),
 connected([VC2,VC3,VC7,VC6]),
 connected([VC11,VC1,VC5]),
 connected([VC9,VC4,VC8]),
 connected([VC10,VC12,GND])
```

Given that omission faults can be compensated using additional actuators, it may be preferable to minimize the potential contribution to commission failure. The respective queries for omission and commission scenarios and the respective fault trees are shown below, see figures 18–21.

Circuit III Omission:

```
?- min_fault_tree(analog_circuit_III_tree(U,1,1,1,1,dark),U,T).
T  =  or(and(fault('switch  1  failed(open)'),  or(fault('switch  3
   failed(open)'),      fault('switch      4      failed(open)'))),
   and(fault('switch    2    failed(open)'),    or(fault('switch    3
   failed(open)'),      fault('switch      4      failed(open)'))),
   fault('battery    6    failed(discharged)'),    fault('lamp    5
   failed(open_circuit)'))
```

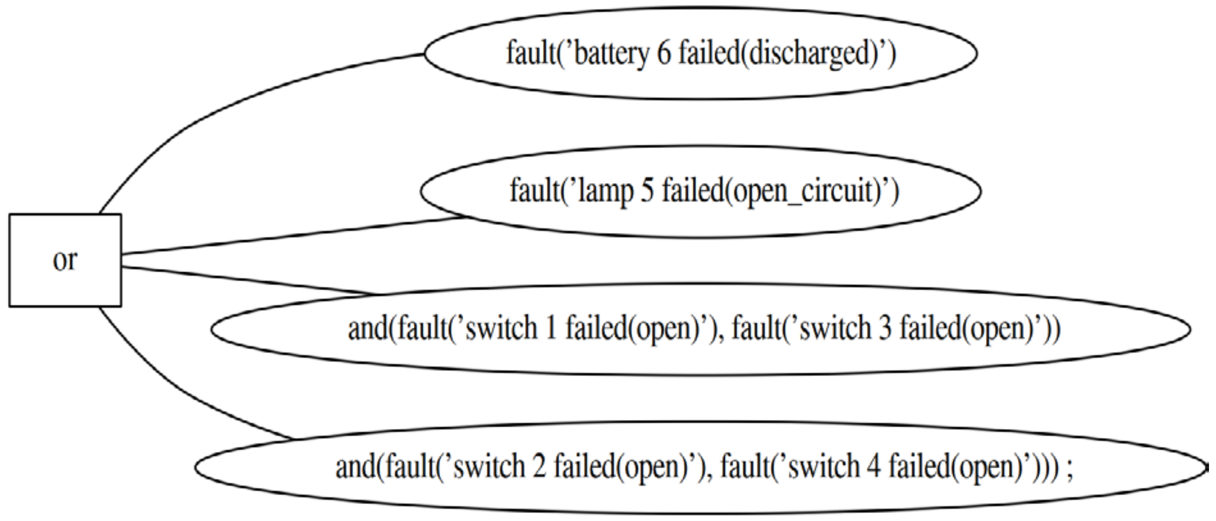

or
fault('battery 6 failed(discharged)')
fault('lamp 5 failed(open_circuit)')
and(fault('switch 1 failed(open)'), fault('switch 3 failed(open)'))
and(fault('switch 2 failed(open)'), fault('switch 4 failed(open)'))) ;

Figure 18. Circuit III omission failure

Circuit III Commission:

```
?- min_fault_tree(analog_circuit_III_tree(U,0,0,0,0,light),U,T).
T  =  or(and(fault('switch  1  failed(short)'),  fault('switch  2
   failed(short)')),    and(fault('switch    3    failed(short)'),
   fault('switch 4 failed(short)'))).......
```

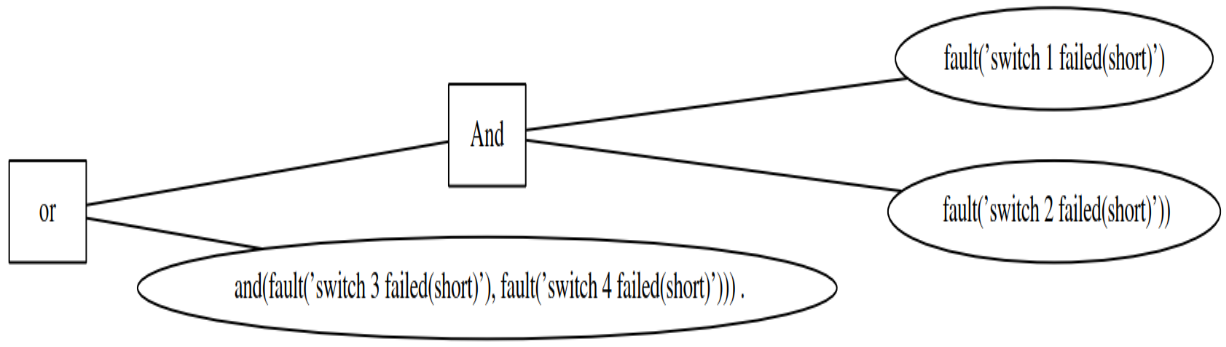

fault('switch 1 failed(short)')
And
or
fault('switch 2 failed(short)'))
and(fault('switch 3 failed(short)'), fault('switch 4 failed(short)'))) .

Figure 19. Circuit III commission failure

Circuit IV Omission:

```
?- min_fault_tree(analog_circuit_IV_tree(U,1,1,1,1,dark),U,T).
T  =  or(fault('battery  6  failed(discharged)'),  fault('lamp  5
   failed(open_circuit)'),  and(fault('switch  1  failed(open)'),
   fault('switch    3    failed(open)')),    and(fault('switch    2
   failed(open)'), fault('switch 4 failed(open)')))
```

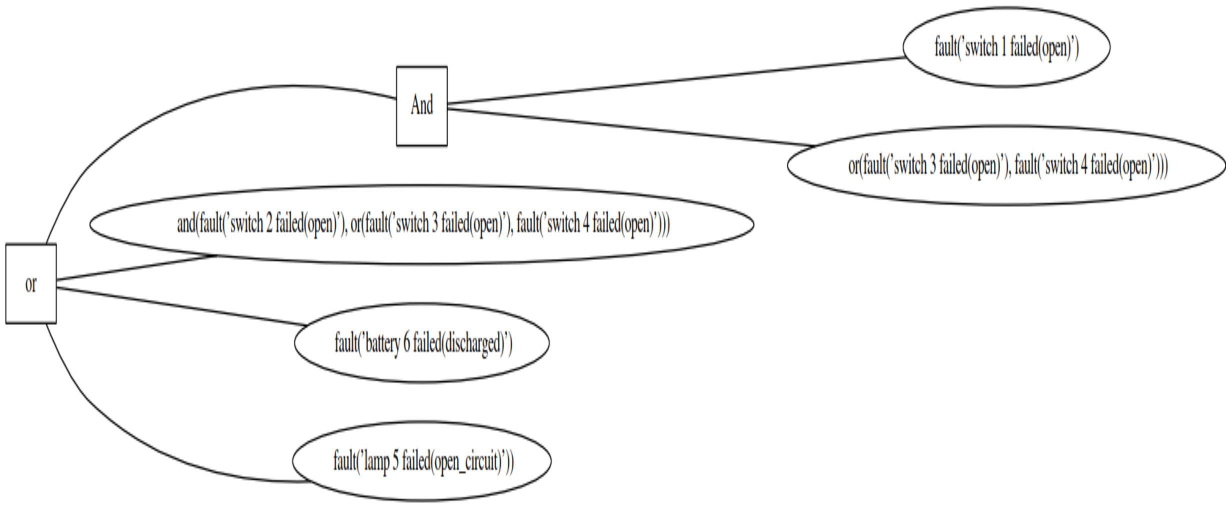

fault('switch 1 failed(open)')
And
or(fault('switch 3 failed(open)'), fault('switch 4 failed(open)'))
and(fault('switch 2 failed(open)'), or(fault('switch 3 failed(open)'), fault('switch 4 failed(open)')))
or
fault('battery 6 failed(discharged)')
fault('lamp 5 failed(open_circuit)')

Figure 20. Circuit IV commission failure

Circuit IV Commission:

```
?- min_fault_tree(analog_circuit_IV_tree(U,0,0,0,0,light),U,T).
T  =  or(and(fault('switch  1  failed(short)'),  or(fault('switch  2
   failed(short)'),     fault('switch     4     failed(short)'))),
   and(fault('switch   3   failed(short)'),   or(fault('switch   2
   failed(short)'), fault('switch 4 failed(short)')))).........
```

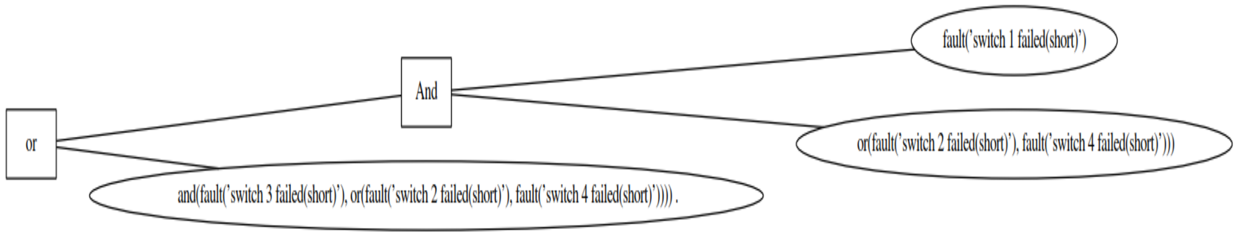

fault('switch 1 failed(short)')
And
or
or(fault('switch 2 failed(short)'), fault('switch 4 failed(short)'))
and(fault('switch 3 failed(short)'), or(fault('switch 2 failed(short)'), fault('switch 4 failed(short)')))) .

Figure 21. Circuit IV commission failure

From these results, the preferred circuit configuration would not include the extra availability cross-strapping because this makes the circuit more vulnerable to commission faults. Note that in the initial exploration, probability numbers are not introduced into the model because the main goal was to investigate the potential to utilize circuit topologies and related behaviors.

10.2.3 Wheel Brake System Case Study

The prior examples show the potential power of the MBSA technique. However, for these to be viable, they need to scale to meet the needs of real systems. As part of this research, this scalability was explored using the context of a wheel brake system example. The circuit behaviors of hydraulic circuits can be similarly captured using analogous abstractions in which pressure replaces voltage and flow replaces current. Some examples of the pump and valve predicates are shown below.

```
pump(hydraulic(V1,I1),hydraulic(V2,I2),Voltage,ok):-
   {I1 = -I2, Voltage=V1-V2,
   V1 >= V2
   }.
pump(hydraulic(V1,0.0),hydraulic(V2,0.0),_Voltage,failed(no_press
   ure)):-
   {V1-V2 = 0.0}.

valve(hydraulic(V1,I1),signal(1),hydraulic(V2,I2),ok) :-
   {I1 = -I2 ,
    I1 = (V2 - V1)/0.001 }.
valve(hydraulic(V1,I1),signal(0),hydraulic(V2,I2),ok):-
   { I1 = -I2, I1 = (V2 -V1)/10000000000}.
valve(hydraulic(V1,I1),signal(_),hydraulic(V2,I2),failed(open)) :-
   {I1 = -I2 ,
    I1 = (V2 - V1)/0.001 }.
valve(hydraulic(V1,I1),signal(_),hydraulic(V2,I2),failed(closed))
   :-
   { I1 = -I2, I1 = (V2 -V1)/10000000000}.
```

However, as the size of the system increases, the state space of exploration grows rapidly. The original wheel brake system comprised 28 components. When one considers that, on average, three failure modes occur per component, this corresponds to 3^28 combinations, which is beyond computational tractability. To this end, learning from the OCRA work explored during the first part of the research, the analysis engine was modified to limit the exploration of concurrent failures. Studies were conducted for two, three, and four concurrent failures. From a probability perspective, such limitation is not unreasonable because, even with component reliability of a failure every 1000 hours, the probability of four concurrent failures is 10E-12.

To limit the number of concurrent failures, a setup predicate was used to generate the database of anticipated concurrent fault manifestations. An excerpt from the database for two concurrent faults is shown below:

```
- dynamic fault_set/1.
fault_set([failed(no_pressure),                          failed(closed),
   failed(stuck_norm_position), ok, ok, ok, ok, ok, ok, ok, ok, ok,
   ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),                          failed(closed),
   failed(stuck_alt_position), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok,
   ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),                            failed(open),
   failed(stuck_norm_position), ok, ok, ok, ok, ok, ok, ok, ok, ok,
   ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),                            failed(open),
   failed(stuck_alt_position), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok,
   ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(closed), ok, failed(closed),
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(closed), ok, failed(open), ok,
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(open), ok, failed(closed), ok,
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(open), ok, failed(open), ok,
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),      failed(closed),      ok,      ok,
   failed(closed), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok,
   ok]).
fault_set([failed(no_pressure), failed(closed), ok, ok, failed(open),
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(open), ok, ok, failed(closed),
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(open), ok, ok, failed(open),
   ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),    failed(closed),    ok,    ok,    ok,
   failed(stuck_on), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),    failed(closed),    ok,    ok,    ok,
   failed(stuck_off), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),    failed(open),    ok,    ok,    ok,
   failed(stuck_on), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),    failed(open),    ok,    ok,    ok,
   failed(stuck_off), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),  failed(closed),  ok,  ok,  ok,  ok,
   failed(stuck_on), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),  failed(closed),  ok,  ok,  ok,  ok,
   failed(stuck_off), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),   failed(open),   ok,   ok,   ok,   ok,
   failed(stuck_on), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure),   failed(open),   ok,   ok,   ok,   ok,
   failed(stuck_off), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
fault_set([failed(no_pressure), failed(closed), ok, ok, ok, ok, ok,
   failed(no_pressure), ok, ok, ok, ok, ok, ok, ok, ok, ok, ok]).
```

The circuit predicate was also modified to reference the fault database, which is passed in a list as shown below:

```
wbs_full_agree_tree_limit([F1,F2,F3,F4,F5,F6,F7,F8,F9,F10,F11,F12,F1
   3,F14,F15,F16,F17,F18,F19,F20,F21,F22,F23,F24],(NormPumpState,
 AltPumpState,
 IsolValveState,
 SelValveState,
 NormLobBcvState,
 NormLibBcvState,
 NormRibBcvState,
 NormRobBcvState,
 NormLobActuatorState,
 NormLibActuatorState,
 NormRibActuatorState,
 NormRobActuatorState,
 AltLeftBcvState,
 AltRightBcvState,
 AltLobActuatorState,
 AltLibActuatorState,
 AltRibActuatorState,
 AltRobActuatorState,
 NormControllerState,
 AltControllerState,
 PLState,
 PRState,
 CPLState,
 CPRState
))),
 PL,PR,CPL,CPR,MECH_CMD,
 ObsNormLob,ObsNormLib,ObsNormRib,ObsNormRob,
 ObsAltLob,ObsAltLib,ObsAltRib,ObsAltRob):-
 component(1,pump(NORM_P_OUT,NORM_P_RTN,3000,F1),NormPumpState),

   component(2,dual_valve(ISO_IN,signal(NormIsoCmd),signal(AltIsoCmd
   ),ISO_OUT,F2),IsolValveState),

   component(3,selector_valve(SEL_NORM_IN,SEL_NORM_OUT,SEL_ALT_IN,SE
   L_ALT_OUT,F3),SelValveState),

   component(4,valve(NORM_LOB_BCV_IN,signal(NormLeftBrakeCmd),NORM_L
   OB_BCV_OUT,F4),NormLobBcvState),

   component(5,valve(NORM_LIB_BCV_IN,signal(NormLeftBrakeCmd),NORM_L
   IB_BCV_OUT,F5),NormLibBcvState),

   component(6,valve(NORM_RIB_BCV_IN,signal(NormRightBrakeCmd),NORM_
   RIB_BCV_OUT,F6),NormRibBcvState),

   component(7,valve(NORM_ROB_BCV_IN,signal(NormRightBrakeCmd),NORM_
   ROB_BCV_OUT,F7),NormRobBcvState),

   component(8,actuator(NORM_LOB_ACT_IN,NORM_LOB_ACT_RTN,ObsNormLob,
   F8),NormLobActuatorState),

   component(9,actuator(NORM_LIB_ACT_IN,NORM_LIB_ACT_RTN,ObsNormLib,
   F9),NormLibActuatorState),

   component(10,actuator(NORM_RIB_ACT_IN,NORM_RIB_ACT_RTN,ObsNormRib
   ,F10),NormRibActuatorState),
   component(11,actuator(NORM_ROB_ACT_IN,NORM_ROB_ACT_RTN,ObsNormRob
   ,F11),NormRobActuatorState),
 component(12,pump(ALT_P_OUT,ALT_P_RTN,3000,F12),AltPumpState),

   component(13,valve(ALT_L_BCV_IN,signal(MECH_CMD),ALT_L_BCV_OUT,F1
   3),AltLeftBcvState),

   component(14,valve(ALT_R_BCV_IN,signal(MECH_CMD),ALT_R_BCV_OUT,F1
   4),AltRightBcvState),

   component(15,actuator(ALT_LOB_ACT_IN,ALT_LOB_ACT_RTN,ObsAltLob,F1
   5),AltLobActuatorState),

   component(16,actuator(ALT_LIB_ACT_IN,ALT_LIB_ACT_RTN,ObsAltLib,F1
   6),AltLibActuatorState),

   component(17,actuator(ALT_RIB_ACT_IN,ALT_RIB_ACT_RTN,ObsAltRib,F1
   7),AltRibActuatorState),

   component(18,actuator(ALT_ROB_ACT_IN,ALT_ROB_ACT_RTN,ObsAltRob,F1
   8),AltRobActuatorState),

   component(19,pedal_sensor(signal(CPL),signal(SENSE_CPL),F19),CPLS
   tate),

   component(20,pedal_sensor(signal(CPR),signal(SENSE_CPR),F20),CPRS
   tate),

   component(21,pedal_sensor(signal(PL),signal(SENSE_PL),F21),PLStat
   e),

   component(22,pedal_sensor(signal(PR),signal(SENSE_PR),F22),PRStat
   e),

   component(23,bcu_agree(signal(SENSE_PL),signal(SENSE_PR),signal(S
   ENSE_CPL),signal(SENSE_CPR),signal(NormLeftBrakeCmd),signal(NormI
   soCmd),F23),NormControllerState),

   component(24,bcu_agree(signal(SENSE_PL),signal(SENSE_PR),signal(S
   ENSE_CPL),signal(SENSE_CPR),signal(NormRightBrakeCmd),signal(AltIsoCmd),
   F24),AltControllerState),
GND_NORM = hydraulic(0,_),
GND_ALT = hydraulic(0,_),
connected([NORM_P_OUT,ISO_IN]),
connected([ISO_OUT,SEL_NORM_IN]),

   connected([SEL_NORM_OUT,NORM_LOB_BCV_IN,NORM_LIB_BCV_IN,NORM_RIB_BCV_IN,N
   ORM_ROB_BCV_IN]),
connected([NORM_LOB_BCV_OUT,NORM_LOB_ACT_IN]),
connected([NORM_LIB_BCV_OUT,NORM_LIB_ACT_IN]),
connected([NORM_RIB_BCV_OUT,NORM_RIB_ACT_IN]),
connected([NORM_ROB_BCV_OUT,NORM_ROB_ACT_IN]),

   connected([GND_NORM,NORM_P_RTN,NORM_LOB_ACT_RTN,NORM_ROB_ACT_RTN,NORM_LIB
   _ACT_RTN,NORM_RIB_ACT_RTN]),
connected([ALT_P_OUT,SEL_ALT_IN]),
connected([SEL_ALT_OUT,ALT_L_BCV_IN,ALT_R_BCV_IN]),
connected([ALT_L_BCV_OUT,ALT_LOB_ACT_IN,ALT_LIB_ACT_IN]),
connected([ALT_R_BCV_OUT,ALT_ROB_ACT_IN,ALT_RIB_ACT_IN]),

   connected([GND_ALT,ALT_P_RTN,ALT_LOB_ACT_RTN,ALT_ROB_ACT_RTN,ALT_LIB_ACT_
   RTN,ALT_RIB_ACT_RTN]),
write('.'
```

The resulting query fault tree for two concurrent failures contributing to an omission failure is shown in figure 22.

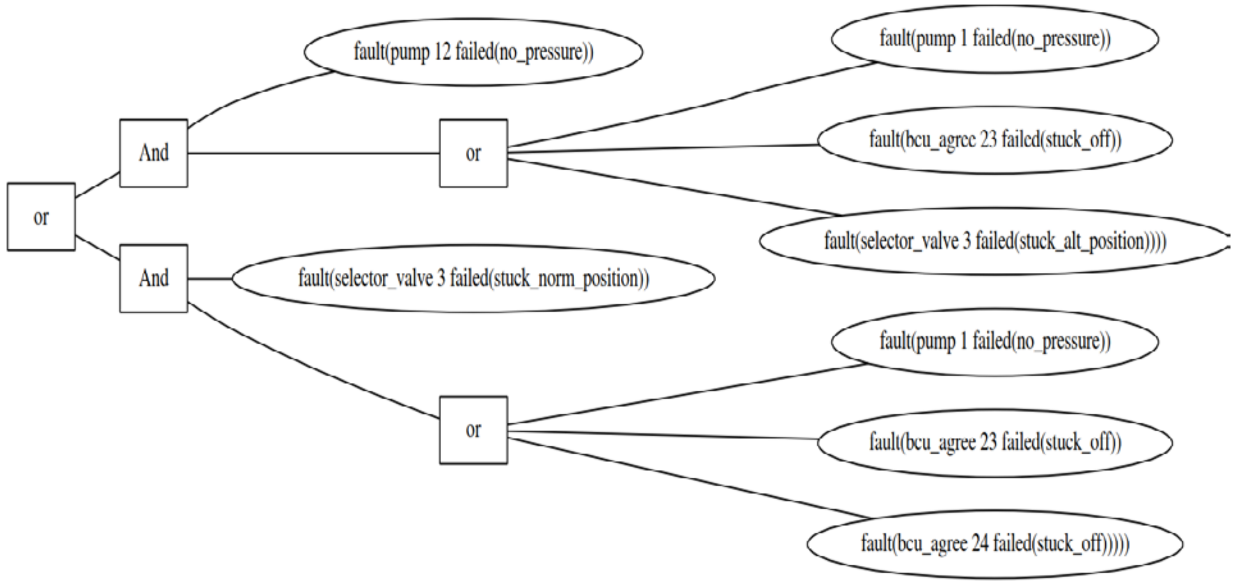

or
And
And
fault(pump 12 failed(no_pressure))
or
fault(pump 1 failed(no_pressure))
fault(bcu_agree 23 failed(stuck_off))
fault(selector_valve 3 failed(stuck_alt_position))))
fault(selector_valve 3 failed(stuck_norm_position))
or
fault(pump 1 failed(no_pressure))
fault(bcu_agree 23 failed(stuck_off))
fault(bcu_agree 24 failed(stuck_off)))))

Figure 22. WBS omission failure limited to two concurrent faults

Additional studies were formed on a reduced brake system model that was explored successfully without limiting the number of concurrent faults. This model was used to explore different pedal-sensing and data-validation braking policies. This allowed rapid feedback with respect to the impact of such policies. However, many of the fault trees generated are too big to include in this document. Figure 23 shows an example fault tree for an omission failure:

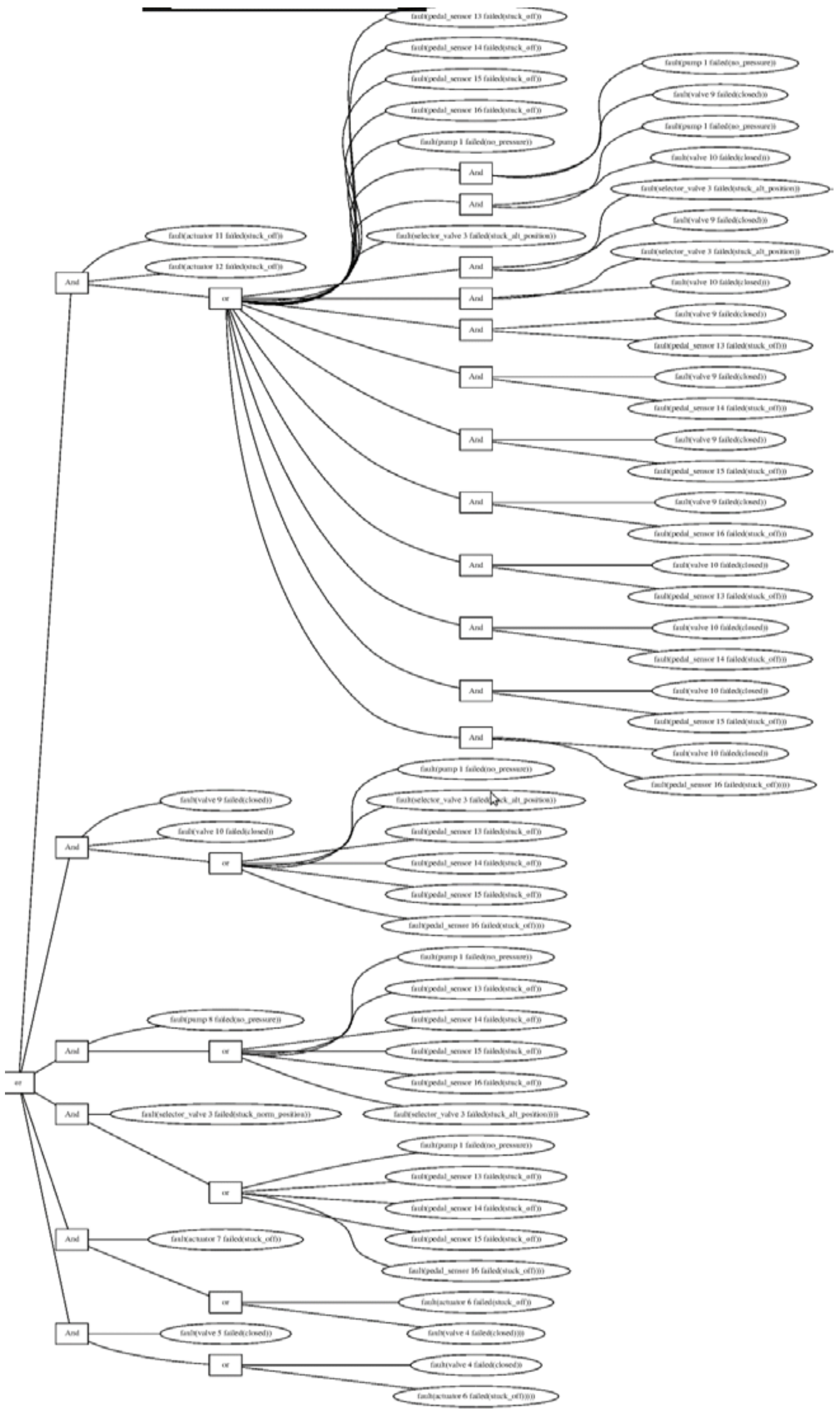


Figure 23. WBS omission failure, unlimited number of faults

The ability to rapidly evaluate the discrete logic of the sensor validation in conjunction with the "circuit aware" behavior of the hydraulics system was very useful. However, in this research, the need for improved state and temporal modeling was also discovered. These findings are summarized in section 12.

10.3 EXPLORING LOGIC-MODEL PROCESSING TO STPA MODELS

10.3.1 Motivation

From the analysis of the incident reports presented during the initial stages of this research program, it was found that many of the incidents could be traced to a misunderstanding of component interactions and failures. STPA [75] may present a valuable framework from which to address such short comings. Therefore, as part of the second phase of research, researchers worked to get a better understanding of the STPA approach. One of the principle advantages of STPA is the flexibility that it affords. It can equally apply and add value at very different levels of abstraction. Recent work has shown such flexibility: analyzing crew procedures [72], human computer interaction [73], and unmanned air vehicles [74]. Relative to the Phase 2 research agenda, the intention was to explore how STPA can be used to support and de-risk the development of complex systems within the context of issues uncovered during the first year's research. As part of this work, researchers are also investigating potential technologies that may assist and augment the STPA analysis. One of the ideas under exploration in this area is the application of logic model processing to assist the hazard and causality analysis. This is the focus of the work presented in this subsection. It is emphasized that this work is at an early stage. It is not the intent to critique STPA, nor point to or assert gaps within the STPA framework. However, it is believed that there may be areas of potential synergy and potential automation and analysis assistance, as the STPA approach is complemented with LMP techniques.

This research does not intend to provide full-scale automation within the context of STPA. One of the main benefits of STPA is the ability to stimulate and unlock the inherent creativity of the human analysts. Therefore, the long-term goal is to support the STPA by providing useful feedback and expert knowledge to further enhance safety-assurance activities. This would be similar to what was demonstrated in the previous section 10.2 for model-based safety fault tree generation.

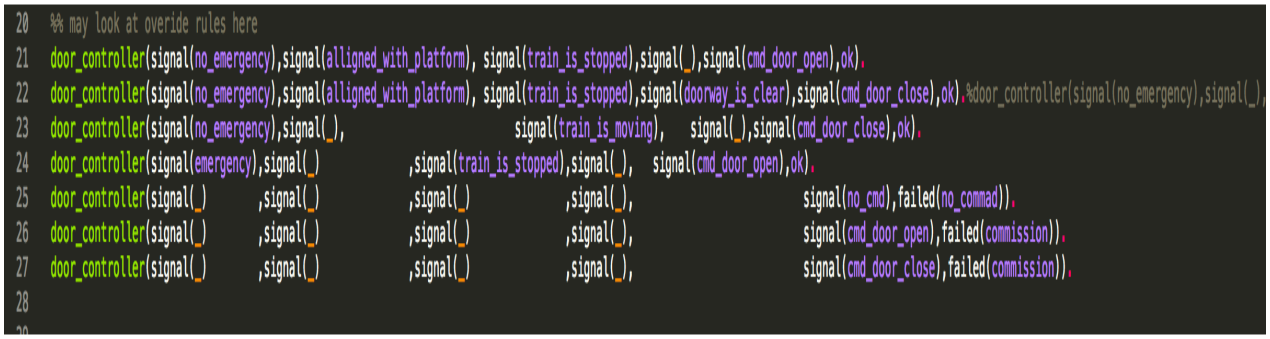
10.3.2 Approach

The context of this work is to explore the potential of applying logical model processing within the context of an STPA-based analysis. An additional goal of this approach is to support the generation of a logical STPA model automatically (i.e., without manual human effort). It is anticipated that the front end of the modeling will be developed within STPA, using tooling such as SAHRA [67] or XSTAMP [32]. For the initial prototyping, the focus has been SAHRA because of its ability to extend the modeling infrastructure using standard SysML constructs or targeted domain specific language extensions, implemented within the Enterprise Architect Model Driven Generation Technology. As discussed previously, part of the initial investigation developed a simple parser that transforms a SAHRA model, exported into XMI format, to a logical representation. Such transformation supports the conversion of the STPA hierarchical control structure and dependencies into logical predicates. For the required behavior and hazard definition, it is anticipated that a constrained natural language may be the preferred approach. This natural language-based approach is inspired by the work of Thomas and Suo [76]. It is also synergistic with related Honeywell internal R&D. The initial constrained natural language approach to specify behavior uses the Easy Approach to Requirements Syntax (EARS)[77]. To evaluate the potential for natural language parsing, the provisions for natural language templates within the Ergo Tool

Suite [78] were briefly explored. This previously had been successfully applied to the capture and representation of financial regulations[79].

For the initial exploration, the simple train-door controller use case from the STPA primer [80] was used. The initial experiments revealed a great deal about the impact of the logical model structure.

In the initial attempt, a similar logical structure that was used for the model-based safety analysis in the previous section was leveraged. Following this approach, the logical predicates representing the system components became flat with large arity[2], as shown below.



```
%% may look at overide rules here
door_controller(signal(no_emergency),signal(alligned_with_platform), signal(train_is_stopped),signal(_),signal(cmd_door_open),ok).
door_controller(signal(no_emergency),signal(alligned_with_platform), signal(train_is_stopped),signal(doorway_is_clear),signal(cmd_door_close),ok).%door_controller(signal(no_emergency),signal(_),
door_controller(signal(no_emergency),signal(_),                     signal(train_is_moving),   signal(_),signal(cmd_door_close),ok).
door_controller(signal(emergency),signal(_)          ,signal(train_is_stopped),signal(_),  signal(cmd_door_open),ok).
door_controller(signal(_)      ,signal(_)          ,signal(_)          ,signal(_),                    signal(no_cmd),failed(no_commad)).
door_controller(signal(_)      ,signal(_)          ,signal(_)          ,signal(_),                    signal(cmd_door_open),failed(commission)).
door_controller(signal(_)      ,signal(_)          ,signal(_)          ,signal(_),                    signal(cmd_door_close),failed(commission)).
```

Such a form requires every predicate to be expressed using all process variables, which may be viewed as suboptimal with respect to how assumed/required behavior would be naturally defined. For example, the requirement below is expressed using a single-process variable, relating to the train speed.

"When train is moving, the controller shall set the door command to close."

Such requirements are often expressed as a single concern and may not require all environment and process variables to be qualified. Therefore, a simpler representation that more closely aligns with the EARS format was sought. A second concern of the logical representation model was the provision to support search-and-finding-system causality relationships. Several experiments were explored here. For the work seeking and investigating the optimal predicate form, encoding of the STPA models was first explored in Prolog because of a greater familiarity with the development environment. An early version of a hand-coded variant for the system composition predicate is shown below. This instance uses variable bindings to implement the causal relationships. Although this form supported provisional analysis, it suffered from suboptimal arity of the predicates discussed previously.

[2] Arity refers to the number of terms in the prolog predicate head.

```
door_system_sim((DC,DA,PD,DS),TrainPos,TrainState,Emergency,DoorCmd,Doorw
   ayState,H):-
    doorway_sensor(signal(DoorwayState),signal(SensedDoorwayState),DS),

   door_controller(signal(Emergency),signal(TrainPos),signal(TrainState)
   ,signal(SensedDoorwayState),signal(DoorCmd),DC),
 door_actuator(signal(DoorCmd),signal(ActuationDoor),DA),
 physical_door(signal(ActuationDoor),signal(DoorPos),PD).
```

The next iteration of the logical model extends the work from Scippacercola [81], originally developed to support an automated FMEA [81]. The work effectively implements lists and list-processing predicates to map the system causality relationships. The advantages of this form over previous attempts are:

1. It removes the need for large arity predicates and supports requirements to be incrementally expressed using a subset of the process variables.
2. It supports a flexible search of all the environmental state variables to assist hazard causality analysis.
3. It implicitly enables component failure and failure modes to be added to the analysis.[3]

```
state(doorway_sensor,doorway_occupied,EV):-
   member([doorway,person_in_doorway],EV).

state(doorway_sensor,doorway_clear,EV):-
   member([doorway,person_not_in_doorway],EV).

state(speed_sensor,sensed_train_is_moving,EV):-
   member([train_speed,train_is_moving],EV).

state(speed_sensor,sensed_train_is_stopped,EV):-
   member([train_speed,train_is_stopped],EV).

state(position_sensor,sensed_alligned_with_platform,EV):-
   member([train_position,alligned_with_platform],EV).

state(position_sensor,sensed_not_alligned_with_platform,EV):-
   member([train_position,not_alligned_with_platform],EV).

state(conductor,prepare_to_depart,EV):-
   member([train_position,alligned_with_platform],EV),
   member([train_speed,train_is_stopped],EV).
```

Following this, the required behavior of the controller is specified. As shown below, this is closely aligned and derived from an EARS requirement expression.

[3] One of the areas of ongoing research is the integration of component failure assumptions within the context of the STPA framework, such as missing, inadequate, incorrect, too early, and too late hierarchal control system interactions.

```
% while emergency exists, when train is stopped, the controller shall
   command the door open
state(controller,open_door,PV):-
   member([conductor,emergency],PV),
   member([speed_sensor,sensed_train_is_stopped],PV).

%while the train is moving the controller shall command the door to close
state(controller,close_door,PV):-
      member([speed_sensor,sensed_train_is_moving],PV),
      member([doorway_sensor,doorway_clear],PV).

%when the conductor is preparing to prepare_to_depart the controller shall
   command door to closed
state(controller,close_door,PV):-
   member([conductor,prepare_to_depart],PV),
```

The hierarchical control structure is shown below. This binds the environmental and process variable lists to the component and state predicates.

```
hcs(EV,PV,CA,OP,P):-
   EV=[[doorway,DS],[train_position,TP],[train_speed,TS],[emergency,EM]]
   ,
   PV=[[doorway_sensor,PV_SDW],[speed_sensor,PV_STS],[position_sensor,PV
   _STP],[conductor,PV_CD]],
   CA=[[door_cmd,DCMD]],
   OP=[[door_actuator,O_MF],[physical_door_position,O_DP]],
   environmental_state(doorway,DS),
   environmental_state(train_position,TP),
   environmental_state(train_speed,TS),
   environmental_state(emergency,EM),
   state(conductor,PV_CD,EV),
   state(doorway_sensor,PV_SDW,EV),
   state(speed_sensor,PV_STS,EV),
   state(position_sensor,PV_STP,EV),
   state(controller,DCMD,PV),
   state(door_actuator,O_MF,CA),
   state(physical_door_position,O_DP,O_MF),
   P is 1.
```

When complete, the model supports the interactive exploration of the hazards in relation to system faults and assumed controller behavior. The exploration is supported by issuing queries as shown below. In this example, none of the variables are grounded. Therefore, the system explores all potential causes. The results indicate that both h2 and h4 are possible.

```
 ?- hcs(EV,PV,CA,OP,P),hazard(H,EV,OP).
EV     =     [[doorway,     person_in_doorway],     [train_position,
   alligned_with_platform],     [train_speed,     train_is_stopped],
   [emergency, active]],
PV    =    [[doorway_sensor,    doorway_occupied],    [speed_sensor,
   sensed_train_is_stopped],                          [position_sensor,
   sensed_alligned_with_platform], [conductor,
   prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close],
   [physical_door_position, closing]],
P = 1,
H = h2 ;
EV = [[doorway, person_in_doorway], [train_position,
   alligned_with_platform], [train_speed, train_is_stopped],
   [emergency, active]],
PV = [[doorway_sensor, doorway_occupied], [speed_sensor,
   sensed_train_is_stopped], [position_sensor,
   sensed_alligned_with_platform], [conductor,
   prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close],
   [physical_door_position, closing]],
P = 1,
H = h4 ;
EV = [[doorway, person_in_doorway], [train_position,
   alligned_with_platform], [train_speed, train_is_stopped],
   [emergency, not_active]],
PV = [[doorway_sensor, doorway_occupied], [speed_sensor,
   sensed_train_is_stopped], [position_sensor,
   sensed_alligned_with_platform], [conductor,
   prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close],
   [physical_door_position, closing]],
P = 1,
H = h4 ;
EV = [[doorway, person_not_in_doorway], [train_position,
   alligned_with_platform], [train_speed, train_is_stopped],
   [emergency, active]],
PV = [[doorway_sensor, doorway_clear], [speed_sensor,
   sensed_train_is_stopped], [position_sensor,
   sensed_alligned_with_platform], [conductor,
   prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close],
   [physical_door_position, closing]],
P = 1,
H = h2 ;
```

A more specific query relating to a selected hazard is also possible (by grounding the hazard identifier as shown below for h2 and h4, highlighted in red).

```
?- hcs(EV,PV,CA,OP,P),hazard(h2,EV,OP).
EV = [[doorway, person_in_doorway], [train_position, alligned_with_platform],
[train_speed, train_is_stopped], [emergency, active]],
PV = [[doorway_sensor, doorway_occupied], [speed_sensor, sensed_train_is_stopped],
[position_sensor, sensed_alligned_with_platform], [conductor, prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close], [physical_door_position, closing]],
P = 1 ;
```

```
EV  =  [[doorway,  person_not_in_doorway],  [train_position,  alligned_with_platform],
[train_speed, train_is_stopped], [emergency, active]],
PV  =  [[doorway_sensor,  doorway_clear],  [speed_sensor,  sensed_train_is_stopped],
[position_sensor, sensed_alligned_with_platform], [conductor, prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close], [physical_door_position, closing]],
P = 1........

?- hcs(EV,PV,CA,OP,P),hazard(h4,EV,OP).
EV  =  [[doorway,  person_in_doorway],  [train_position,  alligned_with_platform],
[train_speed, train_is_stopped], [emergency, active]],
PV  =  [[doorway_sensor,  doorway_occupied],  [speed_sensor,  sensed_train_is_stopped],
[position_sensor, sensed_alligned_with_platform], [conductor, prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close], [physical_door_position, closing]],
P = 1 ;
EV  =  [[doorway,  person_in_doorway],  [train_position,  alligned_with_platform],
[train_speed, train_is_stopped], [emergency, not_active]],
PV  =  [[doorway_sensor,  doorway_occupied],  [speed_sensor,  sensed_train_is_stopped],
[position_sensor, sensed_alligned_with_platform], [conductor, prepare_to_depart]],
CA = [[door_cmd, close_door]],
OP = [[door_actuator, mechanical_force_close], [physical_door_position, closing]],
P = 1
```

Examining the requirements related to hazard 4, it is evident that the requirement to check that the doorway is clear before allowing the close door command is missing.

```
state(controller,close_door,PV):-
   member([conductor,prepare_to_depart],PV).
```

When this is revised as shown below, the subsequent query of the h4 indicates that this hazard is no longer possible.

```
state(controller,close_door,PV):-
   member([conductor,prepare_to_depart],PV),
   member([doorway_sensor,doorway_clear],PV).
```

```
?- hcs(EV,PV,CA,OP,P),hazard(h4,EV,OP).
false.
```

In relation to hazard h2, it is also deduced that there is a missing requirement to open the door when the train is stopped and an emergency condition exists. When such a command is added, as shown below, the resultant query indicates h2 is no longer possible.

```
% while emergency exists, when train is stopped, the controller
   shall command the door open
state(controller,open_door,PV):-
 member([conductor,emergency],PV),
 member([speed_sensor,sensed_train_is_stopped],PV).
```

```
hcs(EV,PV,CA,OP,P),hazard(h2,EV,OP).
false.
```

However, when the unbound query is rerun, it reveals that the added logic also reintroduced some potential hazards.

```
hcs(EV,PV,CA,OP,P),hazard(H,EV,OP).
EV = [[doorway, person_in_doorway], [train_position, alligned_with_platform],
[train_speed, train_is_moving], [emergency, active]],
PV      =      [[doorway_sensor,       doorway_occupied],       [speed_sensor,
sensed_train_is_moving],   [position_sensor,   sensed_alligned_with_platform],
[conductor, emergency]],
CA = [[door_cmd, close_door]],
OP   =   [[door_actuator,   mechanical_force_close],   [physical_door_position,
closing]],
P = 1,
H = h4 ;
EV       =       [[doorway,       person_in_doorway],       [train_position,
not_alligned_with_platform],   [train_speed,   train_is_moving],   [emergency,
active]],
PV      =      [[doorway_sensor,       doorway_occupied],       [speed_sensor,
sensed_train_is_moving], [position_sensor, sensed_not_alligned_with_platform],
[conductor, emergency]],
CA = [[door_cmd, close_door]],
OP   =   [[door_actuator,   mechanical_force_close],   [physical_door_position,
closing]],
P = 1,
H = h4 ;
false.
```

To resolve this, an additional clause is needed to the train-moving predicate, never to close a door if the doorway is not clear.

```
state(controller,close_door,PV):-
 member([speed_sensor,sensed_train_is_moving],PV).
 member([doorway_sensor,doorway_clear],PV).
```

Issuing the unbound query yields “no hazards exist.”

```
?- hcs(EV,PV,CA,OP,P),hazard(H,EV,OP).
false.
```

10.3.3 Natural Language Representation with Ergo

A goal of the translation is to support near-natural language specification of behavioral requirements and hazard. To this end, the Ergo Suite from Coherent Systems was briefly explored. Ergo provides a mechanism that uses templates to assist the binding of natural language and logical predicates. Building from the work implemented in Prolog, simple templates were developed to map EARS-style requirements to the logical representation. For full details of the ErgoText framework and structures, the reader is referred the Ergo Reasoner's manual [82].

```
template(rule,
   \( when the ?sensor1 indicates that ?sensor1_state, the ?controller
   shall command ?command_state \),
            (
            state(?controller,?command_state,?PV):-
            member([?sensor1,?sensor1_state],?PV)
            )

      ).


template(rule,
 \( when the ?sensor1 indicates that ?sensor1_state, and the ?sensor2
   indicates that ?sensor2_state, the ?controller shall command
   ?command_state \) ,
      (
       state(?controller,?command_state,?PV):-
 member([?sensor1,?sensor1_state],?PV),
 member([?sensor2,?sensor2_state],?PV)
      )
   ).
```

Requirement sentences that map to these templates are shown below.

\(when the speed_sensor indicates that train_is_moving, the door_controller shall command door_close \).

\(when the conductor indicates that emergency_is_active, and the speed_sensor indicates that train_is_stopped, the door_controller shall command door_open \).

Hazards can be captured the same way. Below is an example template:

```
template(rule,
 \( hazard ?hazard_id, occurs when the ?object is ?object_state and the
   ?controlled_object is ?controlled_object_state \),
   (
   hazard(?hazard_id,?EV,?OP) :-
 member([?object,?object_state],?EV),
 member([?controlled_object,?controlled_object_state],?OP)
)))
))).
```

Below is the natural language representation:

\(hazard h1, occurs when the train is moving and the door is open \).

Many lessons were learned from this initial experiment. These are presented in section 12 of this report.

10.4 EXPLORING PRINCIPLED ARCHITECTURAL REFINEMENT

STPA presents an improved, more complete, systematic methodology toward hazard elicitation in contrast to processes solely focused on component failure and reliability centric approaches. As such, it may be better suited to address the interaction and related complexities of software-intensive systems. However, STPA currently ceases at Step 2, in which safety requirements and safety constraints are derived at the functional level. Thomas [83] discusses the further refinement of STPA to derive SpecTRM requirements models. However, little detailed work has been published in this area to date. In addition, little published work has been found that targets the architectural refinement of the STPA Step 2 results. This is particularly interesting in relation to the accident survey conducted as part of this research, which uncovered numerous examples of architectural failure. Therefore, one may postulate that such issues would not be addressed by adopting STPA approaches alone. The fact that these failures have also been observed in fielded systems may suggest that current guidance might be lacking in this area.

Because a key focus of this research is to identify gaps with respect to current guidance and emerging safety engineering frameworks, it is believed that addressing the formal architectural refinement is a major need. This is especially important given the growing traction of SPTA, and the potential for it to be adopted as an alternative means of compliance.

The basis of principled architectural refinement is derived from the fail-safe design concept defined in AC-25 1309. The fail-safe design concept requires system designers to address component failures as part of the design process. Specifically, it states:

- In any system or subsystem, the failure of any single element, component, or connection during any one flight should be assumed, regardless of its probability. Such single failures should not be catastrophic.
- Subsequent failures during the same flight, whether detected or latent, and combinations thereof, should also be assumed, unless their joint probability with the first failure is shown to be extremely improbable.

The fail-safe design concept uses the following design principles or techniques to ensure a safe design. The use of only one of these principles or techniques is seldom adequate. A combination of two or more is usually needed to provide a fail-safe design (i.e., to ensure that Major Failure Conditions are Remote, Hazardous Failure Conditions are Extremely Remote, and Catastrophic Failure Conditions are Extremely Improbable [see appendix C C-3]).

(i) Designed Integrity and Quality, including Life Limits, to ensure intended function and prevent failures.
(ii) Redundancy or Backup Systems to enable continued function after any single (or other defined number of) failure(s); e.g., two or more engines, hydraulic systems, flight control systems.
(iii) Isolation and/or Segregation of Systems, Components, and Elements so that the failure of one does not cause the failure of another.
(iv) Proven Reliability so that multiple, independent failures are unlikely to occur during the same flight.
(v) Failure Warning or Indication to provide detection.

(vi) Flight crew Procedures specifying corrective action for use after failure detection.
(vii) Checkability: the capability to check a component’s condition.
(viii) Designed Failure Effect Limits, including the capability to sustain damage, to limit the safety impact or effects of a failure.
(ix) Designed Failure Path to control and direct the effects of a failure in a way that limits its safety impact.
(x) Margins or Factors of Safety to allow for any undefined or unforeseeable adverse conditions.
(xi) Error-Tolerance that considers adverse effects of foreseeable errors during the airplane’s design, test, manufacture, operation, and maintenance

Such considerations may have significant influence on the system architecture and redundancy management. It is believed that this guidance mandating the fail-safe design concept is probably one of the major contributions that has allowed the aerospace industry to achieve its impressive safety record.

It is also believed that STPA, if executed correctly, may yield much better software requirements, to address areas v, vi, and ix. By extending the recent work [73] of integrating human factors within the STPA framework, one may also have the tools to better address the growing complexities of automation surprise and mode confusion. However, the historical positioning of STPA with respect to reliability engineering and probabilistic risk assessment may leave issues related to reliability, architecture, and probabilistic risk assessment to be less explored within the STPA framework. Therefore, it is essential to rectify this, should STPA be under consideration for a potential alternative means of compliance.

Under a safe-design compliant platform, components within this platform should be assumed to fail. Therefore, redundancy and architectural refinement will need to be introduced to support and provide a foundation for the SPTA-derived required behaviors and constraints. Within the conceptual STPA Step 3, the level of architectural redundancy is related to and derived from the hazards and potential unsafe/unwanted and mission control actions. Some initial guidelines are expressed below:

- If there are no unsafe control actions and, subsequently, no unsafe system states, then a simplex design (no redundancy) would be adequate.
  Whatever happens in such a system, no failures of the platform can result in a hazard.
- If there are unsafe control actions (commission failure), then additional hardware redundancy and causality isolation would be required to mitigate these failures. For traditional electromechanical components, such isolation strategies are known (e.g., for actuation, this requires redundant in-series actuation for isolation).
  For processing and sensing subsystems, a fail-stop or fail-op model is required.
  To implement such a model, an architectural-derived requirement to implement separate command and monitoring functions is required to detect and mitigate control platform failures. Should the system have a safe state, then only dual processing and sensing is required (fail-stop).
- Similarly, should there be unsafe control actions that are related to missing control actions, then additional hardware is also required to provide an alternative actuation path in the case of failure. For the actuation function, this would require parallel independent actuating

functions. From a computation and sensing platform perspective, should the system have a safe-state (only availability is needed), then only dual processing and sensing are required.

- If there are unsafe control actions, related to missing or erroneous actuation, then a fail-op (generally quad) actuation architecture is required. From an actuation perspective, this requires two independent lanes of series enforcement. With respect to the compute and sensing architectures, these will also need to fail-operational, given that there is no longer a common fail-safe state.

To show the refinement, a simple, quick example is presented from a hypothetical wheel brake system [84]. Using STPA, one may derive an STPA safety constraint as follows:

"The aircraft must not decelerate after V1."

To avoid negative requirements (which can be a problem to verify) and to better target the requirements for the system of interest, this may be reworded as follows:

"While in takeoff mode and the aircraft speed has exceeded V1, the braking system shall inhibit the application of braking force."

There are several aspects of this requirement that need to be addressed, both from a software functional and architectural perspective. Given such a requirement, from a software functional perspective: Should there be a derived functional requirement to ignore the pilot braking pedals after V1 has been established[4]?

From an architectural perspective, if the software is to be bound to a platform for execution: What platform level fault tolerance is required to mitigate the STPA derived hazards?

Traditionally, in aerospace engineering, the "shall not" constraint constitutes an integrity requirement. That is to say, there is a requirement to isolate and contain commission errors (unwanted control actions). Therefore, given the existence of the "shall not" constraint associated with controller function, to mitigate a single point of failure, the architectural elaboration may refine this control function into command and monitor subfunctions.

In this simple example, the command subfunction activates the braking command, and the monitoring lane mitigates the failure of the command function failure by removing pressure from the actuation circuit via an in-series isolation valve. Therefore, in this simple example, the inhibit requirements are allocated the monitoring subfunction.

However, there are additional attributes that can be introduced by adhering to principled design. Principled design means a design and a design process that incorporates well established (if not

[4] In the Wheel Brake STPA analysis,[87] there were similar hazards related to landing with brakes applied (which could lead to tire burst) and erroneous application of brakes (causing auto-brake disconnect). The analysis concluded that improved annunciation of such conditions is required. However, in addition: Should there be a derived requirement for the braking system to ignore braking commands before ground status (weight on wheels) is confirmed?

widely known) principles that are required for a design to meets its requirements. For the purposes of this research, safety requirements are the primary concern.

Continuing with the brake system example, when the computational functions have been decomposed to replicate the command and monitor subfunctions, it is implicitly known (by principled design) that it is necessary to protect these functions from getting impacted by externally generated errors, including Byzantine errors. Therefore, good design practice would require the command and monitor functions to exchange internal and input state, to ensure that both command and monitoring operational contexts remain congruent.

A review of the accident analysis of the Airbus-320 braking in section 3.1.4 revealed that this exchange was missing. Without this exchange, the electronic control architecture was vulnerable to complete system failure, resulting from the inter-lane disagreement due to short button press. This was the proximate cause of the complete braking system failure described in section 3.1.4. To avoid such issues, an STPA Step 3 is proposed that adds principled design and architecture refinement as the next logical step to follow STPA steps 1 and 2, see figure 24. This Step 3 would provide sufficient guidance and support to ensure such missing derived requirements and architectural blunders are avoided. By integrating the architectural refinement and elaboration within the context of the established steps 1 and 2 of the STPA guided hazard analysis, it is possible to closely tie and document the driving architectural rational with the functional modeling and the driving hazards.

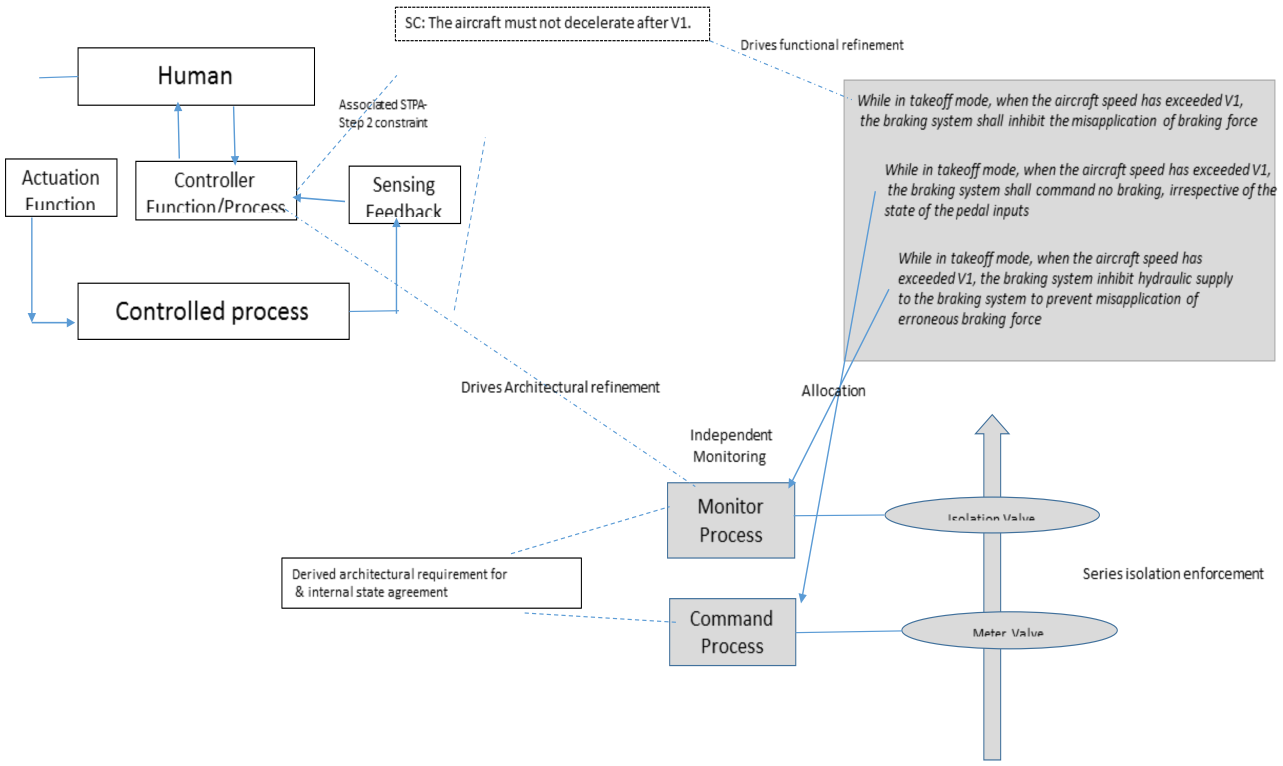

SC: The aircraft must not decelerate after V1.
Drives functional refinement
Human
Associated STPA-Step 2 constraint
Actuation Function
Controller Function/Process
Sensing Feedback
Controlled process
While in takeoff mode, when the aircraft speed has exceeded V1, the braking system shall inhibit the misapplication of braking force
While in takeoff mode, when the aircraft speed has exceeded V1, the braking system shall command no braking, irrespective of the state of the pedal inputs
While in takeoff mode, when the aircraft speed has exceeded V1, the braking system inhibit hydraulic supply to the braking system to prevent misapplication of erroneous braking force
Drives Architectural refinement
Allocation
Independent Monitoring
Monitor Process
Isolation Valve
Derived architectural requirement for & internal state agreement
Series isolation enforcement
Command Process
Meter Valve

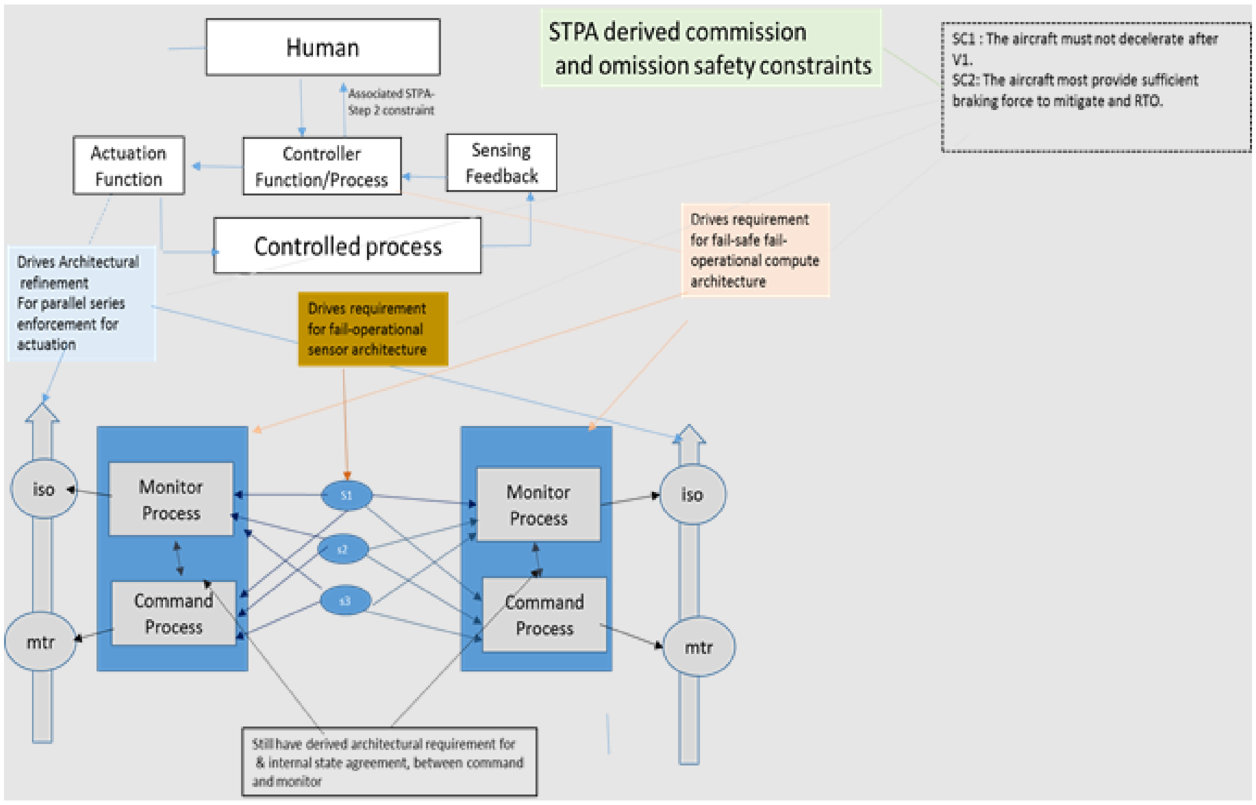
Figure 24. Principled architectural refinement of an STPA unwanted control action

The architectural requirements for the second hazard (i.e., relating a missing control action) can be handled similarly. For example, one would deduce the omission-type safety constraint for the loss of braking during RTO.

"SC2: The aircraft must provide sufficient breaking force to mitigate an RTO."

Adopting the simple principles for safety-directed architectural refinement would cause this requirement to mitigate a missing control action to create the need for two lanes of parallel independent actuation. Similarly, dual independent sensing and compute architectural requirements would be derived to support the required function availability. For brevity, this architecture is not elaborated further here. Instead, only the architectural requirements related to the mitigation of dual unwanted and missing control actions are discussed.

Figure 25 shows the architectural refinement given dual safety constraints, relating potentially unwanted and missing control actions.



Human
STPA derived commission and omission safety constraints
SC1 : The aircraft must not decelerate after V1.
SC2: The aircraft most provide sufficient braking force to mitigate and RTO.
Associated STPA-Step 2 constraint
Actuation Function
Controller Function/Process
Sensing Feedback
Controlled process
Drives requirement for fail-safe fail-operational compute architecture
Drives Architectural refinement For parallel series enforcement for actuation
Drives requirement for fail-operational sensor architecture
iso
Monitor Process
Command Process
mtr
s1
s2
s3
Monitor Process
Command Process
iso
mtr
Still have derived architectural requirement for & internal state agreement, between command and monitor

Figure 25. Principled architectural refinement of unwanted and missing control actions

With respect to the actuation enforcement from the rules above, the system requires two independent channels with series enforcement. However, with respect to the computation and sensing architecture, there are several potential options for sensing and computing architectural refinement. From a requirements perspective, one does not want to overly constrain the design options. However, properties need to be derived (e.g., fail-operation) to ensure adequate redundancy and connectivity to mitigate all faults (including Byzantine manifestations). From understanding fault-tolerance theory, some example refinement guidelines and architectural refinement options can be established:

- If the actuation is dumb (i.e., the output stages have no provisions for sensor voting), the fail-operational platform requirements may be satisfied with quad-computation to mitigate Byzantine failures of computation platform components. The quad computation hardware may be leveraged as a common voting compute platform, or two lanes of independent command monitor enforcement could be used. Applying guidance from principled architectural refinement, the requirements for input and state congruency exchange within the computation function (either between command/monitor lanes or the entire quad computation set) can be formally derived as part of the STPA Step 3 process.
- If, however, the actuation is smart and capable of information exchange and voting, one may mitigate compute failures and lower the requirements (i.e., allow simplex implementation) on the computation architecture alone.

Interestingly, typical triplex schemes without smart actuation (voting) are insufficient, even though they first appear to be “naively fail-op.” A Byzantine member of the triplex can disrupt the set and therefore lead to Byzantine-induced system failure. This was indeed the case for the Space Shuttle [88], which—despite having sufficient quad redundancy and having the reputation of using some of the most dependable software ever written—missed the derived architectural requirements for input and state congruency agreement exchange. Introducing a formal STPA Step 3 would address such design errors by the “principled safety directed rules of architecture.” The rules of architecture (such as the requirement to implement ingress/state exchange) and the independence of power, functional allocation, and common-mode influences would also be implicitly enforced by the guided review frameworks.

Architectural requirements for the sensor function can be derived from the STPA analysis and driving hazards. In the example above, the potential for missing and erroneous control actions related to sensing failure require a fail-operational sensing function. In its simplest form, this may be a triplex sensing implementation, with a mid-value selection function to consolidate the sensing input. In such an arrangement, the architectural refinement rules may require this mid-value selection function to be independently allocated to each of the consuming functions (i.e., each member of the computation platform does its own mid-value selection). An alternative sensing functional architecture may be a dual-dual mapping, in which each command and monitor function simply compares two sets of values and inhibits their respective output in the case of detected sensor disagreement. As stated before, the allocation of the comparison function would be mapped to the command and monitor lanes. Given the requirement for fail-operational sensing, a simplex sensor configuration per lane should be flagged as insufficient if there is a related driving hazard. Ideally, this feedback would be linked to the overlying STPA constraints that drove the architectural requirements.

For all sensing functions, the derived architectural requirements to exchange the sensed input to protect the integrity of the consumers (command and monitor functions) can be easily derived and traced to the safety-related architectural refinement.

There are additional derived requirements that are related to any threshold decisions that may impact the sensing function. Essentially, these relate to waiting for a sufficient period to ensure that the good sensors would agree. During the threshold transition, a Byzantine faulty sensor may be able to force disagreement when working sensors are with the transition zone. Therefore, the command and monitor exchanges need to be cognizant of these issues, and such knowledge would be encoded into the rules of architectural refinement and implicitly fed forward as derived requirements related to sampling and input processing sub-functions.

In the previous initial example, a classical architectural mindset was adopted, which is in line with a literal interpretation of the fail-safe design concept. It may be argued that alternative architectural refinements, requiring fewer sensors or compute redundancy are possible. For example, it may be possible to reduce the required sensor set by leveraging cross-channel comparisons, inline coverage schemes, and predictive model-based arbitration and isolation strategies. However, it is believed that the evaluation of such strategies within the system context established by the STPA derived analysis will bring significant value by enabling a better system informed review of the FDIR logic. It also is important to stress that architecture definition needs to be derived from the upstream functional and safety analysis and should not precede it. Adding error annotations to an architectural model that has not been suitable, derived from an informed safety-directed hazard mitigation process, may not yield the required results. However, by linking an improved "systems thinking based" hazard-analysis framework with formal rules of architectural refinement, the resulting integrated framework will guarantee that safety-related constraints and requirements are met with the implemented architectural mitigation strategies. As STPA steps 1 and 2 are principled on a control theoretic approach to the elicitation of hazards, the proposed extension of Step 3 would be a suitably principled and informed method to support architectural refinement of the upstream STPA-derived requirements and constraints.

There are several other aspects relating to architectural FDIR scrubbing and the requirement to detect latent failures that can also be principally derived and linked to the principles and requirements for the fail-safe design concept. Because of time restrictions, these have not been fully explored under the current research task. However, should the community agree to the value of the proposed Step 3 of the STPA analysis, such issues would be addressed as the detailed requirements and methodology for Step 3 are evolved and finalized. The key principle of the proposed extension is to link the architectural requirements to the safety constraints and the requirements from the upstream safety analysis while simultaneously supporting the refinement with knowledge of best-design practices and dependable system theory. With the vision of the knowledge-based assisted system introduced in section 9, such rules of principled architectural refinement may be incorporated within the logical databases and, therefore, used to support automated architectural synthesis/ or architectural validation.

10.5 PRINCIPLED ARCHITECTURAL REFINEMENT EXAMPLE

To show the power of principled architectural refinement, one may revisit the simple train door controller use case from the STPA primer [80] and ask the questions: How many door-is-closed sensors are needed to satisfy the following door-system requirements?

1. The door system must operate correctly, given any single door-system failure.
2. Door system must not disable train for a single fault.
3. Door system must tell train not to move if door is not closed.
4. Passengers must be able to exit train in an emergency.

The first thing that most designers would notice is conflicting requirements. What if there is an emergency and the train is moving? STPA would find this in Step 1. This leads to a derived requirement:

5. The train must stop in an emergency.

This requirement must be satisfied outside of the door system and then is used as an assumption by the door system.

The answer to the number of sensors question is six. This would be a surprise to most designers and to all of the authors who have written papers about this problem. However, a principled architectural refinement would find that two doors are needed to satisfy requirements 1 and 4. It also would find that the door-is-closed sensing on each door needs to be available to satisfy requirements 2 and 3, and needs integrity to satisfy requirements 1 and 3. Then, a principled architectural refinement would find that triplex sensors would be needed on each door to satisfy the availability and integrity requirements. Two doors with three sensors each totals six sensors.

11. SAFETY MANAGEMENT AND CHANGE IMPACT ANALYSIS

Effective change management is paramount to life-cycle system safety. Task 3 focused on identifying the gaps with respect to the current certification guidance and the needs of complex systems. To address this task, the team reviewed the current guidance governing change management. The related internal Honeywell procedures that addressed the certification guidance were then analyzed, and the incremental changes and updates that were made to these procedures were specifically reviewed, in accordance with the Honeywell adoption of the SMS. A brief description of the SMS is attached as appendix A.

Some initial brief observations on the application of the SMS principles to the current certification guidance are presented in this section.

As a member state of ICAO, the FAA has adopted SMS as an approach, in part, to manage safety risk for a variety of entity types (e.g., operators, airports and others).

Current Advisory Circular (AC): AC 21.101-1B establishes the certification basis of changed aeronautical products. This AC provides guidance for the application of the Changed Product Rule, Title 14 of the Code of Federal Regulations (14 CFR) §§ 21.101 and 21.19, for changes made to type certificated aeronautical products. Generally, 14 CFR 21.101b is related to onboard equipment systems and equipment. Under 21.101b, there is no process regarding how to manage changes to onboard equipment to which integrated systems is a major part. Unfortunately, it is difficult to reconcile the hundreds of requirements to ensure they are complete, systems get modified, upgrades occur, certain components become obsolete and are replaced with new equipment, and new software is added. In addition to CPR, Instructions for Continued Airworthiness (ICA is applied if maintenance procedures to correct and put products back into service assuming original certification basis are unchanged). Current CPR processes, continued airworthiness processes, SAE Aerospace Recommended Practice (ARP) 4754A for establishing a development assurance process, and SAE ARP 4761, describing a safety assessment process, which is a supporting part of the larger development process described by ARP 4754A, all do not adhere to the fundamental principles of a safety management as related to change.

All processes within design and manufacturing must recognize all four components of a safety-management system interact as part of all established processes. For each process, each SMS component must be addressed. One such process is management of change. Regardless of the size of the change, under SMS principles, all components must be applied. As required by 14 CFR Part 5, any time there is a new system, a change to the system, a change in a procedure, or recognition of a safety risk through the continuous monitoring and safety-assurance process, a safety risk assessment (SRA) must be performed. An SRA is an SMS process in itself.

For the purpose of this research to demonstrate the gap in current processes and a full functioning SMS, the need for an SRA will be based on recognition of a safety risk through the continuous monitoring and safety-assurance process. If an SRA is required, the safety-risk-assessment process must address all four components. For example, if a safety hazard is identified, the Safety Risk Management component will address it through system design, hazard identification, analysis, risk assessment, and identifying risk-mitigation strategies or controls for the proposed change. Safety assurance will be applied within the SRA process to determine if the applied controls or risk-mitigation strategies are working as expected. It will continuously monitor new systems (i.e., collect data to analyze), review the risk mitigations implemented, identifying if risk-mitigation strategies are working as intended, if strategies were properly implemented, and to identify any unintended consequences or new hazards. Safety promotion with communication is required to be developed and maintained in a formal way that explains why a change was made while providing training commensurate with the position(s) for the changes made. Last, safety promotion applies the safety culture into the very fabric of the organization and addresses an environment that encourages a learning, reporting, flexible, informed, and just culture. Finally, safety policy will be addressed that will identify who is accountable to accept the residual risk after controls have been determined and where change will be documented.

Under design and manufacturer design-change processes, the Safety Assessment Reviews resemble a strong safety program that identifies hazards and addresses the hazards with a particular risk mitigation or control but does not apply the remaining four components, resulting in multiple gaps.

In review of documentation from a design and manufacturing company, components of an SMS and significant gaps exist together. “The purpose of the upfront urgent action outlined by this process is proactive strategy to assure safety or potential safety issues are not overlooked, minimized, or inappropriately prioritized. The objective is to minimize any real or perceived risk to the fleet with a fully coordinated investigation and analysis as warranted.”

The current procedure completes part of the SMS process. After product release, hazards are identified. The hazards or potential safety issues reported are presented to the experienced supervisor and the board, who determine if the hazard is a safety issue and, if so, who must be included in the process (i.e., FAA, OEM, or company providing Type Certificate or STC). It is proactive in an, SRA process but not in a safety assurance process.

In a further effort to compare and contrast SMS, the following charts (see figure 26) were modified to include the SMS in relation to the ARP 4761 Safety Assessment Process and the STPA Hazard Analysis Process.

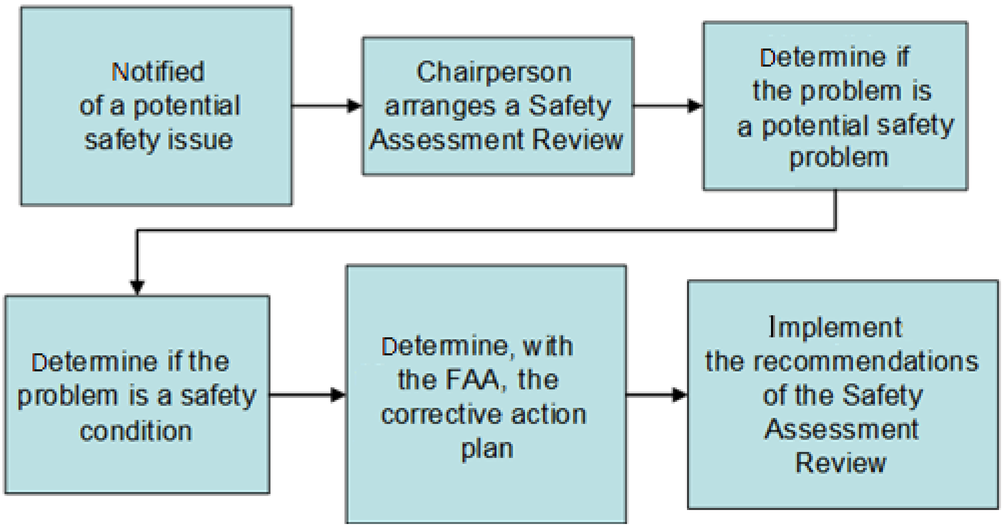

Notified of a potential safety issue
Chairperson arranges a Safety Assessment Review
Determine if the problem is a potential safety problem
Determine if the problem is a safety condition
Determine, with the FAA, the corrective action plan
Implement the recommendations of the Safety Assessment Review

Figure 26. High-level change process model

In addition to the continued integration of all four components of an SMS into each process, the most prevalent differences and recommendations for civil aircraft design and manufacture organization can be summarized in the following items:

1. Processes must be re-engineered to include the four components of an SMS.
2. Trending data must be obtained from products currently in service that have not generated a safety report or hazard report. A culture of continuous monitoring must be established. While removing faults and generating requirements during design reduce risk effectively, it is not always attainable. Continuous monitoring must be planned and measured to capture unintended and unplanned consequences that may occur from mechanical and human factors considerations.
3. Safety culture must be incorporated into the organization. Design and manufacturer companies must begin by establishing a baseline safety culture. This can be accomplished with a safety culture survey done at least on a yearly basis. From that baseline, the company must continue measuring the culture while incorporating the four components into all processes.
4. An SMS would require that the risk-mitigation plan include processes to continually monitor new systems (i.e., collect data to analyze), review the risk mitigations implemented, and identify if risk mitigations strategies were working as intended and were properly implemented, and to identify any unintended consequences or new hazards.
5. This process seems to help decrease risk by better design, but does not take into account the fact that with the interaction of man, machine, media, management, and mission, there will be unintended consequences regardless of design. This leads to a more robust safety assurance to be required by an SMS.
6. STPA is a process that would fit under Safety Risk Management of an SMS but not safety assurance. Even with the review of previous accidents, it is being done as a reactive risk-mitigation strategy. An SMS would focus more on proactive and predictive strategies to reduce risk while also using reactive strategies. STPA documentation indicates "After identifying the accidents, the system safety hazards to be considered, and the system-level

safety requirements (constraints), the next step in STPA is to create a model of the aircraft functional control structure."

12. RESEARCH LESSONS LEARNED AND RECOMMENDATIONS

The pace of technology innovation is currently fueling a relentless increase in system complexity. New applications, increasing connectivity, and new levels of autonomy are enabling new application areas that were unimaginable a few years ago. However, with each new technology and application, the ability to assure system safety becomes increasingly difficult. Although the systems-thinking approach to safety may better address the increasing software content and increasing dynamics of human-system and system-system interactions, more is needed to mitigate the inherent weakness of human reasoning [73].

In section 9, a vision of an integrated assistance system, built on logical model-processing technologies, is presented. Although this was an audacious goal, given the relatively small remaining research funds, the investigations and experiments performed during the last third of this research effort show promise in many areas.

The logical models explored herein are too simplistic and crude for real systems. This work shows that better provisions, state, and time are needed. If more time had been available, the back-end logical abstractions would have been revised to use better logical processing technologies, such as coinductive logic programming [87] and related techniques targeted towards cyber-physical systems [88]. These approaches support the reasoning of infinite and cyclic systems, and better address the complexities of real-world cyber-physical control systems. In addition, these modeling approaches and backend reasoning systems also better address "real time" and do not require artificial discretization strategies. The ability to integrate related temporal reasoning with the front-end STPA functional modeling and the back-end implementation modeling may be greatly beneficial because temporal emergent properties are a common source of system failures.

12.1 OBSERVATIONS WITH RESPECT TO CERTIFICATION GUIDANCE

A key goal of this research has been to identify gaps and potential areas where the current certification guidance may be improved. The following sections capture observations and recommendations founded on many hours and conversations made throughout the course of the research program. These sections comprise the results of discussions among many product-assurance and product-design groups within Honeywell, which include the perceptions of development risks and opportunities.

12.1.1 Culture and Clarifying Systems Versus Software Responsivities

One of the surprising areas omitted from the current certification guidance is the subject of safety culture. There is little material and guidance specifically targeting safety culture and safety-culture maturity within a guidance framework. Yet, within the SMS framework, it is evident the establishment and maturation of a proactive safety culture is one of the fundamental goals of an SMS. In large part, it is the safety culture established with an SMS that provides the foundation for the wider SMS initiatives and activities. Therefore, it is recommended that additional guidance be created to track and measure the maturity of development safety cultures. This would be a beneficial extension to the current certification guidance framework. The need for such guidance seems to be getting more important as organizations evolve to incorporate increasingly globalized and outsourced engineering teams. Understanding the rationale and intent behind the design-assurance activities appears to be fundamental to their effectiveness. As organizations mature their internal processes, there is a risk of a "check-the-box" attitude emerging, as design assurance activities are relegated to "certification must do" activities, which are seen as costs to be endured rather than tools to facilitate the creation of safe designs. Developing a more in-depth understating of the whys for these activities can be an important contribution to countering such verification mindset fatigue. In contrast to a few decades ago, when the entire team would be consolidated and largely assigned to a project throughout the development lifecycle, today's development teams may involve several groups, each focused on a specific area, or just a portion of a product's lifecycle. Such limited roles and exposure to the full product and design assurance context can significantly impact the appreciation of the principles and "safety mindset" that provide the foundation of effective design assurance. Similarly, additional frameworks to communicate the system-safety implications of software need to be established. A software engineer, experiencing firsthand the first power-on of an APU in a test cell next door, may quickly and easily develop an inherent understanding of the properties and risks related to high-speed rotational machinery and the importance of the safety-related shutdown logic. However, getting an equivalent appreciation of such properties to his/her counterpart working solely with UML diagrams 8000 miles away is not easy. In complex integrated systems, for which the platform development may be largely unaware of the hosted applications' designs, developing a holistic understanding of the system-safety considerations can be harder still.

Some of the risks related to such partitioned teams may be mitigated through the use of model-based design approaches that support the integration of multiple system perspectives into a common model. As discussed by Rierson [89], the model-based approaches can improve communication and support earlier and improve simulation-based support of requirements validation.

However, such approaches may also incur risks as the traditionally separate levels of the design hierarchy, such as system and software engineering, are compressed. For example, consider an OEM presenting Simulink diagrams as software high-level requirements to a supplier. Often, with such approaches, the requirement intent above the Simulink diagram may be omitted, leaving downstream supplier-implementation teams limited information to gauge the suitability of the Simulink implementation. As Rierson [89] points out, this reduces the potential for collateral requirements validation that was implicit within traditional non-model-based approaches, Although DO-331 attempts to address such issues by requiring higher-level requirements to be established above such models, it largely addresses the issues from a software perspective, leaving a lot of ambiguity with respect to system versus software responsibilities. This is shown by the assumed system and software life-cycle processes of DO-331 in figure 27.

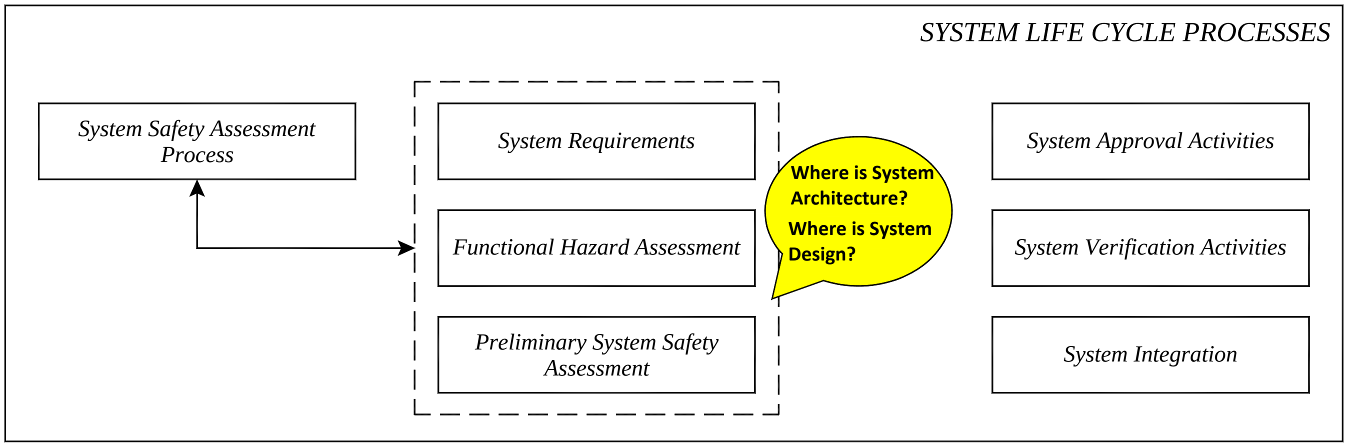

SYSTEM LIFE CYCLE PROCESSES
System Safety Assessment Process
System Requirements
Functional Hazard Assessment
Preliminary System Safety Assessment
Where is System Architecture?
Where is System Design?
System Approval Activities
System Verification Activities
System Integration

Figure 27. Annotated from MB-2.1, information flow between system and software life cycle processes of DO-331

It is interesting that neither the system architecture nor the system-design models are acknowledged with this process flow. This omission may be due to the software-centric context of the supplement. However, improved guidance should be created to clarify how such model-based design workflows can be executed across systems/software and OEM/supplier boundaries, that can satisfy the objectives of both ARP-4754A and DO-331. Introducing formal objectives for system architecture and system-design models may provide great benefit. Further recommendations are given in section 12.1.3

12.1.2 Certification Guidance Complexity and Assurance Cases

In prior research, Goosen [90] made many recommendations to simplify the structure of the certification guidance material. Although this research was focused on the complexities and difficulties related to multitier developments, many of these findings are reiterated here. An improved understanding of the certification guidance material will assist all participants to better understand context and rationale for their respective activities.

To mitigate the complexity of the assurance guidance for complex systems, Kritzinger [91] suggests that formal safety assurance cases should be developed using structured notations, such as GSN [92]. Recent research by Knight [93] summarizes how such approaches can introduce additional benefits over traditional approaches. This report agrees with these findings and recognizes the benefits that such approaches add to the certification dialog. Such approaches support a more holistic understanding of the system safety rationale and how the certification guidance has been interpreted and discharged by the applicant. Although such measures are not precluded by the current certification guidance, they are not formally encouraged. As demonstrated from the Nimrod experience [94], the use of such approaches does not necessarily guarantee safety. Therefore, additional guidance to develop criteria and to establish frameworks to formally leverage such approaches within the certification context may be beneficial. The improved guidance targeting the applicability of assurance cases within the traditional design-assurance framework may also provide an incremental stepping stone toward the goals of the "Overarching Property"-based approaches to certification currently under discussion [95].

However, in prior NASA-funded research [96], risks related to the graphical-only notations regarding assurance case notations were identified, such as GSN. As assurance cases become large and complex, graphical notations may become unwieldy and difficult to review. In prior research, answer set and logical-model processing techniques were applied to published GSN-based safety cases. Through the application of this logical approach, inconsistencies and missing evidence were quickly identified. In separate research, Rushby [97] also discusses the benefits of mechanized assisted assurance reasoning support. Given the anticipated increasing complexity of the next generation of systems, it is believed that more research focused in this area would be particularly beneficial. Of interest is the role and qualification of potential tooling and automation assistance strategies in this area. Therefore, further research into machine-assisted reasoning of assurance cases is recommended as a potentially beneficial area of future study. Applying the technology of the Inquire textbook [62] to the large corpus of certification guidance may also greatly mitigate the complexity challenges of this information set.

12.1.3 Formal Architectural Modeling, Fault Specification, and Architectural Metrics

An interesting observation related to ARP 4754A is the notion of architecture. The term architecture appears 77 times throughout the document. However, the term "architecture" is not formally defined in the document definitions. In section 4.1.6, the development of the system architecture is summarized as follows:

> The system architecture is accomplished as a part of this process activity. This system architecture establishes the structure and boundaries within which specific item designs are implemented to meet all of the established safety and technical performance requirements. The outputs of this activity include the system architecture to the item level and the allocation of system function and safety requirements to the appropriate items.

Given the importance of the system architectural argumentation with respect to system safety objectives, the informal treatment of the system architectural definition is not as useful as it should be. One may question if the missing definition of architecture targets functional, logical, or physical architectural attributes, or all three.

Within the model-based development community, multiple solutions [98, 99] are targeting more formal definitions of architecture and architectural refinement. Standardized notations to capture and represent software and platform deployment architectures [16] are also rapidly maturing. Extensions to these languages that integrate different aspects, such as error modeling [100] or IMA-based networking [101], also are under development. With respect to certification guidance, it would not be correct for 4754A to call out specific modeling languages or tools. However, drawing from the experience of the wider community, identifying the required attributes for functional, logical, and physical architectural representations may be greatly beneficial. Of interest, relative to the findings during the first portion of this research, is the definition of system interface and interface assumptions, especially regarding the assumed failure modes. It is believed that a more explicit definition in this area will greatly assist system review, especially in development scenarios with distributed teams. A suitably formal definition may also enable systemwide consistency checking and case-based reasoned test generation, as discussed in the previous section. However, at a minimum, the guidance should require sufficient formal architectural definition to support a more thorough and formal review of the system architectural assumptions and claims with respect to the anticipated faults and fault propagation[5].

In this regard a more formal specification of fault and fault propagation assumptions with respect to the architectural policies and fault-mitigation and fault-mitigation strategies would be greatly beneficial. Guidance should not mandate specific languages or tools that should be used, but instead clarify the level of definition and specification that are needed to reduce the architectural risks. In this regard, there is a large body of knowledge and research within the model-based system (AADL/SysML) engineering communities, which can be harvested. However, from review of the current approaches, it is believed that better integration is needed between the error and nominal behavioral perspectives. In AADL, these two perspectives are currently addressed by independent annexes, with little integration and provisions to check the consistency between these perspectives. From the brief experimentation adopting logical model processing, a more integrated 'physics aware' approach appears to present good potential.

Therefore, from a certification perspective, additional guidance is needed to ensure sufficient validation of model-based safety abstractions. Additional research, targeted to derive the requirements and criteria to validate model-based safety abstractions with respect to the physical systems that they represent, is also recommended.

12.1.3.1 Extending Hazard Analyses and Functional Decomposition Using STPA

With respect to functional modeling and functional decomposition, control-centric modeling (as prescribed by STPA) would be preferable to the traditional analytical decomposition prescribed by the current guidance.

[5] It is also emphasized that such information would be implicitly generated, using the principle architectural refinement proposed in a previous section.

It is believed that this control-centric view of the STPA hierarchical control structures is more inclusive than the strict analytical reduction illustrated in the AIR. Great benefit is seen in extending the hazard-identification process outlined in the current guidance with the STPA approaches. To this end, additional criteria to access and gauge quality and applicability of STPA model artifacts should be generated. It is understood that at the time of this writing, an AIR to support the deployment of STPA is under preparation. This may be a good start toward suitable guidance. However, contrary to some suggestions, it is not believed that STPA alone can be a viable alternative to 4754A and 4761. A blended strategy that leverages the best parts of both current and STPA approaches is believed to yield the most benefit. It is also believed that additional guidance to more formally address the architectural refinement of systems underdevelopment is needed for both traditional and STPA-based hazard analyses. Automated assistance to support the manual STPA processes may also introduce good value.

The integration of such models and perspective within the larger context of integrated model-based system engineering workflows is a potentially fertile area for future research.

12.1.3.2 Change Management

From mapping the impact of implementing the SMS toward Honeywell's internal problem-resolution and change-management processes, it was found that many of the documents that aided Honeywell with respect to this process refinement may not be widely known to the general community. Only one of the documents [102] referenced in the internal procedures has been officially released. Honeywell's knowledge of the related documents was largely a function of the work close dialog with the FAA with respect to SMS adoption. A greater awareness of some of these supporting documents [103–106] may be very beneficial to the wider industry.

12.1.3.3 Requirements Management

Requirement errors are at the root of integration safety issues. The guidance on requirements validation in ARP-4754A is sparse. Some improvement around validation and consistency checking would be valuable, preferably using automated means with guaranteed coverage.

Formal requirements analysis (e.g., assume/guarantee) is suggested for validation of the functional requirements (from the high-level functional specification) and the safety requirements (from HAZOP or STPA) of an initial architecture. After refinement of the architecture for hazard elimination or mitigation, HW/SW functional allocation and specification are performed. The detailed design and verifications follow, using the validated requirements.

The varying definitions of fault, failure, and error used by different parts of the industry have significant potential for creating confusion and communications barriers. It is recommended that they be unified in line with the IFIP definitions.

Avionics are complex systems that cannot be comprehended in their entirety. Complexity is manageable but only if abstracted to representational models at multiple levels of detail and analysis methods appropriate to that level of detail applied. Current text-based requirements capture has limitations with respect to comprehending the complexity of interactions between requirements clauses. This is also true when systems are modified and unintended side effects occur.

Every level of requirements analysis should depend on predecessors forming a repeatable and traceable hierarchy.

12.1.3.4 TSO-Related Observations

TSO products sometimes or mostly lack the FMEA data needed to safely integrate them into larger systems. It is recommended that TSO products used in a safety-critical part of a larger system should be treated in the same way as TC/STC/PMA products, taking the larger system view and using the same analysis techniques. Where this data is not available, the integrator should backfill this deficiency and take appropriate mitigation steps in the system design.

The safety issues associated with complex digital systems integration have been identified to be mainly centered on the problem of escapes of inadequate system level and component requirements rather than the implementation and verification of the requirements. Accordingly, this report describes a methodology based on defining the high-level architecture and component requirements in a formal language and subjecting them to validation using formal model-checking techniques. This approach is rigorous in the sense that incomplete or inconsistent requirements are detected by the model-checking process prior to implementation and show unequivocally that component requirements satisfy the system-level requirements.

13. FUTURE WORK

One of the major findings, as described in section 10.4, is the need for principled design and architecture refinement, potentially as an STPA Step 3. Such a capability is needed to create safe designs as systems and systems of systems become more complex. This capability is also needed to combat the lack of sufficient expertise among current and possible future system designers, particularly as complexity drives increasing "stovepiping" of ever more narrow disciplines and the concomitant increase in the number of specialized disciplines required to create a system. Orthogonal to stovepiping is the increasing need to use multiple levels and types of abstraction to make facets of a complex design comprehensible. The cross product of stovepiping and multiple abstraction layers means an even greater increase in the number of interfaces among designers and the potential for "cracks" in the design-information flow that can lead to design errors and oversights.

Incorporating a knowledge-centric and logic-model processing approach within the development process may introduce great value. The integrated reasoning of "what we is built", together with "how it was built" will also provide a solid foundation for the overall assurance case argumentation.

Tools and processes must be created to implement this principled design and architecture-refinement process to avoid the repetition of known architectural design errors.

14. REFERENCES

1. RTCA Report. (2011). "Software Considerations In Airborne Systems And Equipment Certification,". DO-178".

2. RTCA Report. (2000). "Design Assurance Guidance For Airborne Electronic," DO-254.

3. ATSB Report. (2011). “In-flight upset 154 km west of Learmonth, WA, 7 October 2008, VH-QPA, Airbus A330-303 Flight QF72 (Final).” (AO-2008-070).

4. BEA Report. (2012). “Final Report - on the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro – Paris.” (BEA f-cp090601).

5. ATSB Report. (2007). “In-flight Upset Event 240km North-West of Perth, WA, Malaysian Airlines Flight 124, Boeing B777-200 on 1 August 2005.” (Aviation Occurrence Report – 200503722).

6. Dutch Safety Board Report. (2010). “Crashed during approach, Boeing 737-800, near Amsterdam Schiphol Airport, 25 February 2009.” (https://www.onderzoeksraad.nl/en/page/1182/turkish-airlines-crashed-during-approach-boeing-737-800-amsterdam)

7. Comisión de Investigación de Accidentes e Incidentes de Aviación Civil, “Accident occurred on 21 May 1998 to Aircraft Airbus A-320-212 Registration G-UKLL At Ibiza Airport , Balearic Islands,” 1998-05-21-ES.

8. AAIB Report. (2007). “Report on the incident to Virgin Atlantic Airbus A340-642, registration G-VATL en-route from Hong Kong to London Heathrow on 8 February 2005.” (EW/C2005/2/3).

9. KNKT Report. (2015). “NTSC Final Investigation Report into Airbus 320-216 - PK-AXC involving Indonesia Air Asia Company Flight QZ-8501 during flight at Karimata Strait, Coordinate 3˚37’19‘S-109˚42’41’E - Java Sea on 28 December 2014.” (KNKT.14.12.29.04).

10. The Ministry of Transport of Japan Aircraft Accident Investigation Commission Report. (1994). “China Airlines Airbus A300B4-622R, Nagoya Airport, Nagoya, Japan, April 26, 1994.”

11. NTSB Report. (2007). “Factual Report - Predator B UA Crash Nogales, AZ, 25th April 2006.” (CHI06MA121).

12. BEA Report. (2010). “Accident on 27 November 2008 off the coast of Canet-Plage to the Airbus A320-232 registered D-AXLA operated by XL Airways Germany.” (d-la081127).

13. South African Civil Aviation Authority, South African Incident Investigation Division (AIID) Report. (2010). “Boeing B747-400, G-BYGA, Group ‘A’ Leading edge slats retracted on take-off roll – Tambo Airport, SA. – 11 May 2009.” (CA18/3/2/0717).

14. BEA Report. (1994). “REPORT on the incident on 24 September 1994 during approach to Orly to the Airbus A 310 registered YR-LCA operated by TAROM.” (YR-A940924A).

15. Gelman, G., Feigh, K., and Rushby, J. (2014). “Example of a complementary use of model checking and human performance simulation.” *IEEE Trans. Human-Machine Syst.*, *44*(5),

576–590.

16. SAE Standard AS-5506/B. (2004). "Architecture Analysis & Design Language (AADL)," SAE, Warrandale, PA.

17. Feiler, P. H. and Gluch, D. P. (2012). "*Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*," (1st Ed.). Boston, MA: Addison-Wesley Professional.

18. SAE Standard AS5506/3 Draft 0.98 (2014). "Architecture Analysis and Design Language (AADL) Annex Volume 3: Annex E: Error Model Language," SAE, Warrandale, PA.

19. Ellidiss Technologies (2011). "*COMPASS Graphical Modelling User Manual*."

20. Bozzano M., Cimatti A., Katoen JP., Nguyen V.Y., Noll T., Roveri M. (2009) "The COMPASS approach: Correctness, modelling and performability of aerospace systems." *SAFECOMP 2009*, 173–186.

21. Nguyen, V. Y. (2013). "*Trustworthy Spacecraft Design Using Formal Methods*," (PhD Thesis). no. October, p. 196.

22. RWTH Aachen University (2012). "*COMPASS Integrated Platform User Manual*," (Report D9).

23. RWTH Aachen University (2013). "*Semantics of the COMPASS System-Level Integrated Modeling (SLIM) Language*."

24. RWTH Aachen University (2013). "*Specification of the COMPASS System-Level Integrated Modeling (SLIM) Language."*

25. Guiotto, A., De Ferluc, R., Bozzano, M., Cimatti, A., Gario, M., and Yushtein, Y. (2014). "Fame process: A dedicated development and V&V process for FDIR." *European Space Agency, (Special Publication), SP 725*(1).

26. Mahadevan, N., Abdelwahed, S., Dubey, A., and Karsai, G. (2010). "*Distributed diagnosis of complex systems using timed failure propagation graph models*," *AUTOTESTCON, 2010 IEEE*, (1–6).

27. Bouissou M. (2005). "Automated Dependability Analysis of Complex Systems with the KB3 Workbench : the Experience of EDF R & D."

28. Batteux, M., Prosvirnova, T., Rauzy, A., and Kloul, L. (2013). "The AltaRica 3.0 project for model-based safety assessment." *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, (741–746).

29. Cavada, R. *et al.* (2014). "The nuXmv Symbolic Model Checker." Biere, A., and Bloem, R., (Eds.), *Computer Aided Verification, vol. 8559* (334–342), New York, NY: Springer International Publishing.

30. Fondazione Bruno Kessler Embedded Systems Unit. (2012). "*XSAP User Manual*."

31. Bozzano, M. *et al.* (2015). "Contract-based design and safety analysis of an aircraft wheel brake system: Revisiting AIR-6110 with formal methods," Trento, Seattle, WA: Fondazione Bruno Kessler; The Boeing Company.

32. Abdulkhaleq, A. and Wagner, S. (2015). XSTAMPP : "An eXtensible STAMP Platform As Tool Support for Safety Engineering." *STAMP Conference.*, (2–5).

33. Garrett, C. J., Guarro, S. B., and Apostolakis, G. E. (1995). "The Dynamic Flowgraph Methodology for Assessing the Dependability of Embedded Software Systems." *IEEE Transactions on Systems, Man, and Cybernetics*, *25*(5), 824–840.

34. Larson, B.R., Chalin, P., Hatcliff, J. (2013) BLESS: "Formal Specification and Verification of Behaviors for Embedded Systems with Software."Brat, G., Rungta, N., Venet, A. (eds) *NASA Formal Methods. NFM 2013. Lecture Notes in Computer Science, vol 7871*. (276–290). Berlin, Heidelberg: Springer.

35. B. Larson, "Behavior Language for Embedded Systems with Software Annex Sublanguage for AADL," Sep. 2014

36. Larson, B.R., Zhang, Y., Barrett, S.C., Hatcliff, J., Jones, P.L. (2015). "Enabling Safe Interoperation by Medical Device Virtual Integration." *IEEE Design & Test*, *32*(5), 74–88.

37. Object Management Group (2010) "SysML-Modelica Transformation Specification," no. June, p. 105, 2010.

38. SAE Standard ARP-4754A (2010). "Guidelines for Development of Civil Aircraft and Systems," SAE, Warrendale, PA.

39. SAE Standard ARP-4761 (1996). "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," SAE, Warrendale, PA.

40. Micouin, P. (2014). "*Model Based Systems Engineering: Fundamentals and Methods*." Hoboken, NJ: Wiley Publishing.

41. Gacek, A., Backes, J., Whalen, M., and Cofer, D. (2013). "*AGREE Users Guide v0.4*."

42. Leveson, N.G. (2011). "*Engineering a Safer World: Systems Thinking Applied to Safety*." Cambridge, MA: MIT Press.

43. Bozzano M., *et al*. (2015) "Formal Design and Safety Analysis of AIR6110 Wheel Brake System." Kroening D., Păsăreanu C. (eds) *Computer Aided Verification. CAV 2015. Lecture Notes in Computer Science, vol 9206.* (518–535). Cham, Switzerland: Springer International Publishing.

44. Zimmerman, M., Leveson, N. G., and Lundqvist, K. (2002). "Investigating the readability of state-based formal requirements specification languages." ICSE '02 Proceedings of the

24th International Conference on Software Engineering, Pages 33-43, Orlando, FL — May 19 - 25, 2002, ACM New York, NY, USA ©2002

45. Leveson, N., "A new accident model for engineering safer systems." *Safety Science, 42*(4), 237–270.

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

47. Leveson, N. G., Reese, J. D., & Heimdahl, M. P. E. (1998). "SpecTRM: A CAD system for digital automation." AIAA/IEEE Digital Avionics Systems Conference - Proceedings (Vol. 1). IEEE.

48. Leveson, N.G. (2000). "Intent Specifications: An Approach to Building Human-Centered Specifications." *IEEE Transactions on Software Engineering, 26*(1), 15–36.

49. Dulac, N., Viguier, T., Leveson, N., and Storey, M.-A. (2002) "On the use of visualization in formal requirements specification," 2002, pp. 71–80.

50. SAE Standard AIR-6110. (2011). "Contiguous Aircraft/System Development Process Example," SAE, Warrendale, PA.

51. Bozzano M., Cimatti A., Fernandes Pires A., Jones D., Kimberly G., Petri T., Robinson R., Tonetta S., "Formal Design and Safety Analysis of AIR6110 Wheel Brake System," Paper presented at, CAV 2015: 27th International Conference on Computer Aided Verification, July 18-24, 2015, San Francisco, California [Online]. Available: https://es-static.fbk.eu/projects/air6110/index.php?n=Main.Download.

52. Cimatti, A. and Tonetta, S. (2015). "Contracts-refinement proof system for component-based embedded systems." *Science of Computer Programming, 97*(P3), 333–348.

53. FAA Order 8150.1C, Technical Standard Order Program, (2012).

54. FAA. (2010, Feb.) Advisory Circular AC21-46. *Technical Standard Order Program*, (AC21-46). Washington, D.C.: Government Publishing Office.

55. FAA Order TSO-C87a, Technical Standard Order: Airborne Low-Range Radio Altimeter, (2012).

56. EuroCAE Standard ED-30. (1980). "Minimum Performance Specification for Airborne Low-Range Radio (Radar) Altimeter Equipment," EuroCAE, Paris, France.

57. Hall, B., Driscoll, K., and Schweiker, K., "Verification and validation of flight critical systems (VVFCS)." In *Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st*, (1–18).

58. SEI Report. (2014). "AADL Fault Modeling and Analysis Within an ARP4761 Safety

Assessment.” (CMU/SEI-2014-TR-020).

59. Feiler, P. and Delange, J. (2016). “Automated Fault Tree Analysis from AADL Models,” *ACM SIGAda Ada Letters Archive, 36*(2), 39–46.

60. Hailey, P. (2013). “*Anybody can contribute knowledge using KnowBuddy*.” [PowerPoint slides]. Retrieved from http://haleyai.com/documents/AnybodyKnowBuddy.pptx.

61. Chaudhri, V. K., Clark, P. E., Mishra, S., Pacheco, J., and Spaulding, A. (2004). “AURA : Capturing Knowledge and Answering Questions on Science Textbooks ∗ Overall Design and Requirements Analysis,” *Artif. Intell.*, 2004.

62. Chaudhri, V. K. *et al.* (2013). “Inquire Biology : A Textbook that Answers Questions.” *AI Magazine, 34*(3). 55–72.

63. Mavin, A. and Wilkinson, P. (2010). “BIG EARS (The return of ‘Easy Approach to Requirements Syntax’),” In *Proc. 2010 18th IEEE Int. Requir. Eng. Conf. RE2010*, vol. 6, (277–282).

64. Ligęza A., Potempa T. (2014) “AI Approach to Formal Analysis of BPMN Models: Towards a Logical Model for BPMN Diagrams.”Mach-Król M., Pełech-Pilichowski T. (eds) Advances in Business ICT. Advances in Intelligent Systems and Computing, vol 257. Springer, Cham (69–88).

65. Dissaux P., Hall B., “Merging and Processing Heterogeneous Models.” 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016, TOULOUSE, France. (Document Number: hal- 01291342). Available: https://hal.archives-ouvertes.fr/hal-01291342

66. Dissaux P., Farail P., “Model Verification: Return of experience,” 6th European Congress on Embedded Real Time Software and Systems (ERTS 2014), February 2014, TOULOUSE, France. Available: https://www.researchgate.net/publication/260981935_Model_Verification_Return_of_experience

67. SAHRA. (n.d.). Available: http://www.sahra.ch/.

68. Sparx Systems. (2016). “Enterprise Architect.” [Computer software].

69. OMG. (n.d.). “Object Management Group Business Process Model and Notation.” Available: http://www.bpmn.org/.

70. Störrle, H. (2007). “A PROLOG-based approach to representing and querying software engineering models,” In *CEUR Workshop Proceedings.*, vol. 274, (71–83).

71. Almendros-Jimenez, J. M. and Iribarne, L. (2013). “*A Model Transformation Language based on Logic Programming.*” From the 39th International Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic.

72. Scarinci, A. (2017). “Monitoring safety during airline operations : a systems approach.” (Thesis). Massachusetts Institute of Technology, Department of Aeronautics and Astronautics. Retrieved from http://hdl.handle.net/1721.1/112479.

73. France, M. E. (2017). “Engineering for Humans : A New Extension to STPA.” (Thesis). Massachusetts Institute of Technology, Department of Aeronautics and Astronautics. Retrieved from http://hdl.handle.net/1721.1/112357.

74. Bagschik, G., Stolte, T., and Maurer, M. (2017). “Safety Analysis Based on Systems Theory Applied to an Unmanned Protective Vehicle.” *Procedia Engineering, 179*, 61–71.

75. Krauss, S. S., Rejzek, M., and Hilbes, C. (2015). “Tool Qualification Considerations for Tools Supporting STPA.” *Procedia Engineering, 128*, 15–24.

76. Thomas J., Suo D., “STPA-based Method to Identify and Control Feature Interactions in Large Complex Systems,” 3rd European STAMP Workshop, STAMP EU 2015, Procedia Engineering, Volume 128, 2015, Pages 12-14

77. Mavin, A., Wilkinson, P., Harwood, A., and Novak, M. (2016). “EARS (Easy Approach to Requirements Syntax),” In *Proceedings from the IEEE International Conference Requirements Engineering*, (317–322).

78. Oherent Knowledge. (n.d.). “Ergo Suite Platform.” Available: http://coherentknowledge.com/product-overview-ergo-suite-platform/.

79. Grosof, B. *et al.* (2015). “Automated Decision Support for Financial Regulatory / Policy Compliance , using Textual Rulelog,” no. May, pp. 1–8.

80. Levenson N., Thomas J., “STPA Handbook,”, March 2018, Published by “Partnership for Systems Approaches to Safety and Security (PSASS)”, Available: http://psas.scripts.mit.edu/home/materials/

81. Scippacercola, F., Pietrantuono, R., Russo, S., Silva, N. P., and Federico, N. (2015). “SysML-based and Prolog-supported FMEA,” In *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).* (174–181).

82. Coherent Knowledge. (2017). “*ERGO Reasoner User’s Manual*.” Mercer Island, WA: Coherent Knowledge Systems.

83. Thomas, J. and Suo, D. (2015). “STPA-based Method to Identify and Control Feature Interactions in Large Complex Systems.” *Procedia Engineering, 128,* 12–14.

84. Leveson N., Wilkinson C., Fleming C., Thomas J., Tracy I., “A Comparison of STPA and the ARP 4761 Safety Assessment Process, MIT Technical Report,” June 2014, Available: http://sunnyday.mit.edu/papers/ARP4761-Comparison-Report-final-1.pdf

85. Driscoll K.R., Hall B., Schweiker K., “Application Agreement and Integration Services,” NASA/CR-2013-217963, NF1676L-15890, February 1, 2013

86. Miller, G. A. (1956). “The magical number seven, plus or minus two: some limits on our capacity for processing information.” *Psychological Review, 63*(2), 81-97.

87. Simon, L. E. (2006). “Extending Logic Programming with Coinduction.” (Dissertation). Presented to the Faculty of the Graduate School of The University of Texas at Dallas

88. Saeedloei, N. (2011). “Modeling and Verification of Real-time and Cyber-Physical Systems.” (Dissertation). University of Texas.

89. Rierson, L. (2013). “*Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance 1*.” Boca Raton, FL: CRC Press.

90. NASA Report. (2014). “Regulatory Compliance in Multi-Tier Supplier Networks.” (NASA/CR-2014-218550).

91. Kritzinger, D. (2016). “*Aircraft system safety: Assessments for initial airworthiness certification*.” Sawston, UK: Woodhead Publishing.

92. Kelly T., Weaver R., “The Goal Structuring Notation – A Safety Argument Notation,” Journal Proceedings of the dependable systems and networks 2004 workshop on assurance cases, 2004/7/1, Available: https://www-users.cs.york.ac.uk/tpk/dsn2004.pdf

93. NASA Report. (2017). “Understanding What It Means for Assurance Cases to “Work.”” (NASA/CR–2017-219582).

94. Haddon-Cave C., “An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 Aircraft XV230 in Afghanistan in 2006,” Ordered by the House of Commons (Great Britain), 28 October 2009

95. Holloway C.M., ”Understanding the Overarching Properties: First Steps,” FAA Technical Report Draft, September 2018

96. “Certware.” [Online]. Available: https://nasa.github.io/CertWare/

97. Rushby, J. (2014). “*Mechanized support for assurance case argumentation.*” Paper presented at the 1st International Workshop on Argument for Agreement and Assurance (AAA 2013), Kanagawa Japan, October 2013. Appears in Springer LNAI Vol. 8417, pp. 304–318.

98. Ansys. (n.d.). SCADE Architect. Available: http://www.esterel-technologies.com/products/scade-architect/scade-avionics-package/.

99. PolarSys. (n.d.). Capella. Available: www.polarsys.org/capella/.

100. Larson, B., Hatcliff, J., Fowler, K., and Delange, J. (2013). “Illustrating the AADL error modeling annex (v.2) using a simple safety-critical medical device.” *ACM SIGAda Ada Letters*, *33*(3), 65–84.

101. Robati, T., El Kouhen, A., Gherbi, A., Hamadou, S., and Mullins, J. (2014). "*An extension for AADL to model mixed-criticality avionic systems deployed on IMA architectures with TTEthernet.*" Presented at the 1st Workshop on Architecture Centric Virtual Integration @ the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014), Sep 2014, Valencia, Spain. CEUR-WS, 1233, pp.14-27.

102. FAA. (2008, September). Advisory Circular 39-8. "*Continued Airworthiness Assessments of Powerplant and Auxiliary Power Unit Installations of Transport Category Airplanes.*" (AC 39-8). Washington, D.C.: Government Publishing Office.

103. FAA, *Draft TARAM Handbook.*

104. FAA, "Small Airplane Risk Analysis (SARA) Handbook." (2010).

105. FAA, "CONTINUED AIRWORTHINESS ASSESSMENTS OF POWERPLANT AND AUXILIARY POWER UNIT INSTALLATIONS OF TRANSPORT CATEGORY AIRPLANES", Advisory Circular 39-8(9/8/03).

106. FAA, "Rotorcraft Risk Analysis Handbook." (9/15/14).

APPENDIX A—SAFETY MANAGEMENT SYSTEMS OVERVIEW

The implementation of a safety management system (SMS) represents a fundamental shift in the way an aviation organization does business. In effect, safety becomes an integral part of the everyday operations of the organization and is no longer considered an adjunct function belonging to the safety office.

An SMS requires organizations to adopt and actively manage their business and safety objectives and incorporate them into their everyday organization practices and culture.

The word system means “to bring together or combine.” This is not a new term. The philosopher Aristotle first identified systems. An SMS involves the transfer of some of the responsibilities for aviation safety issues from the regulator to the individual organization. This is a role shift in which the regulator oversees the effectiveness of the SMS but withdraws from day-to-day involvement in the organizations it regulates. The day-to-day issues are discovered, analyzed, and corrected internally by the organizations.

From the organization’s perspective, the success of the system hinges on the development of a safety culture that promotes open reporting through non-punitive disciplinary policies and continual improvement through proactive safety assessments and quality assurance.

The SMS philosophy requires that responsibility and accountability for safety be retained within the management structure of the organization. Management is ultimately responsible for safety and other aspects of the enterprise. The responsibility for safety, however, resides with every member of the organization. In safety management, everyone has a role to play. Regardless of the size and complexity of an organization, senior management will have a significant role in developing and sustaining an organization’s safety culture. Without the sincere, unconditional commitment of all levels of management, any attempt at an effective safety program will be unsuccessful. Safety management requires the time, financial resources, and consideration that only senior management can provide.

Some examples of management commitment and support may include discussing safety matters as the priority during staff meetings, participating in safety committees and reviews, allocating the necessary resources—such as time and money—to safety matters, and setting a personal example. However it is manifested, the importance of support from management cannot be underestimated.

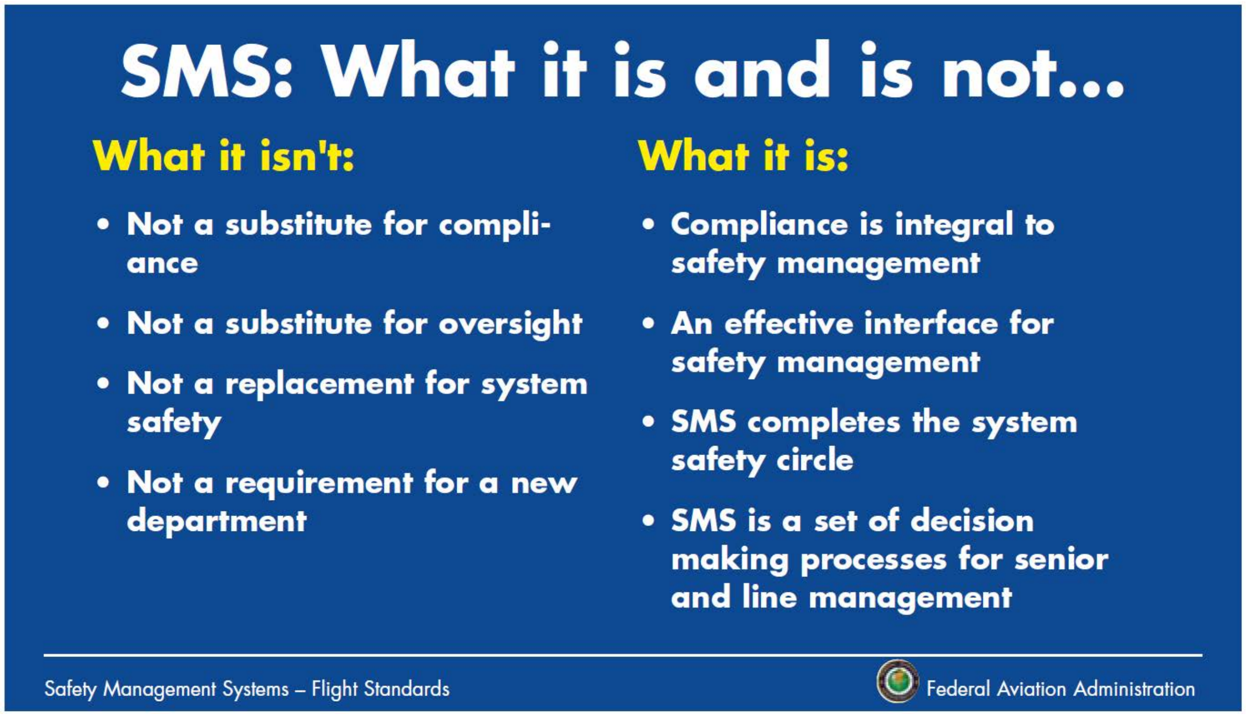
Why Is an SMS Needed?

An SMS facilitates the proactive identification of hazards, promotes the development of an improved safety culture, modifies the attitudes and behavior of personnel to prevent damage to aircraft or equipment, and makes for a safer workplace. An SMS helps organizations avoid wasting financial and human resources and management’s time from being focused on minor or irrelevant issues.

An SMS allows employees to create ownership of the organizational processes and procedures to prevent errors. An SMS lets managers identify hazards, assess risks, and build a case to justify controls that will reduce risk to acceptable levels. An SMS is a proven process for managing risk that ties all elements of the organization together, laterally and vertically, and ensures appropriate allocation of resources to safety issues. An SMS provides an organization with the capacity to anticipate and address safety issues before they lead to an incident or accident. An SMS also provides management with the ability to deal effectively with accidents and near misses so that valuable lessons are applied to improve safety and efficiency. The SMS approach reduces loss and improves productivity.

Definition of an SMS

An SMS is defined as a coordinated, comprehensive set of processes designed to direct and control resources to optimally manage safety. An SMS takes unrelated processes and builds them into one coherent structure to achieve a higher level of safety performance, making safety management an integral part of overall risk management. An SMS is based on leadership and accountability. It requires proactive hazard identification, risk management, information control, auditing, and training. It also includes incident and accident investigation and analysis. Figure A-1 contrasts the attributes of a successful SMS versus the attributes of a good safety program.



SMS: What it is and is not...

What it isn't:

- Not a substitute for compli-ance
- Not a substitute for oversight
- Not a replacement for system safety
- Not a requirement for a new department

What it is:

- Compliance is integral to safety management
- An effective interface for safety management
- SMS completes the system safety circle
- SMS is a set of decision making processes for senior and line management

Safety Management Systems – Flight Standards

Federal Aviation Administration

Figure A-1. Attributes of an SMS, FAA

Safety management is woven into the fabric of an organization. It becomes part of the culture and affects how people do their jobs. The organizational structures and activities that make up a safety-management system are found throughout an organization. Every employee contributes to the safety health of the organization. In some organizations, safety-management activity will be more visible than in others, but the system must be integrated into “the way things are done.” This will be achieved by the implementation and continuing support of a safety program based on coherent policies and procedures.

The Accountable Executive

One person must have the responsibility to oversee SMS development, implementation, and operation. This person is called the accountable executive. The accountable executive must be the "champion" for the SMS program. The managers of the line operational functions, from middle management to frontline managers and supervisors, manage the operations in which risk is incurred. These managers and supervisors are the key safety personnel of the SMS. For each process, the element that defines responsibilities for definition and documentation of aviation safety responsibilities, applies to all components, elements, and processes.

Key Safety Personnel

Top management has the ultimate responsibility for the SMS and should provide the resources essential to implement and maintain the SMS. Top management should appoint members of management who, irrespective of other responsibilities, have responsibilities and authority including:

- Ensuring the processes needed for the SMS are established, implemented, and maintained.
- Ensuring the promotion of awareness of safety requirements throughout the organization.
- Ensuring that aviation safety-related positions, responsibilities, and authorities are defined, documented, and communicated throughout the organization.

The Four Components, or Pillars, of an SMS:

The ICAO Document 9859 and FAA Advisory Circular 120-92A state that the SMS is structured on four basic components, sometimes called pillars, of safety management:

- Safety Policy
- Safety Risk Management
- Safety Assurance
- Safety Promotion

Safety Policy

Every type of management system must define policies, procedures, and organizational structures to accomplish its goals. An SMS must have policies and procedures in place that explicitly describe responsibility, authority, accountability, and expectations. Most importantly, safety must be a core value.

The safety policy should state that safety has a very high priority within the organization. It is the accountable manager's way of establishing the importance of safety as it relates to the overall scope of operations. Leadership sets the tone. Senior management's commitment will not lead to positive action unless commitment is expressed as direction. Management must develop and communicate safety policies that delegate specific responsibilities and hold people accountable for meeting safety performance goals.

The policy must be clear and concise, and must emphasize top-level support, including a commitment to:

- Implementing an SMS.
- Continuously improve the level of safety.
- Manage safety risks.
- Comply with applicable regulatory requirements.
- Encourage, not retaliate against, employees who report safety issues.
- Establish standards for acceptable behavior.
- Provide management guidance for setting and reviewing safety objectives.
- Provide documentation.
- Communicate with all employees and parties.
- Provide periodic review of policies to ensure they remain relevant and appropriate to the organization.
- Identify the responsibilities of management and employees with respect to safety performance.
- Integrate safety management with other critical management systems within the organization.
- Include safety components to all job descriptions that clearly define the responsibility and accountability for each individual within the organization.

Safety Risk Management

A formal system of hazard identification and management is fundamental in controlling an acceptable level of risk. A well-designed risk-management system describes operational processes across department and organizational boundaries, identifies key hazards and measures them, methodically assesses risk, and implements controls to mitigate risk.

Understanding the hazards and inherent risks associated with everyday activities allows the organization to minimize unsafe acts and respond proactively by improving the processes, conditions, and other systemic issues that lead to unsafe acts. These systemic/organizational elements include training, budgeting, procedures, planning, marketing, and other organizational factors known to play roles in many systems-based accidents. In this way, safety management becomes a core function and is not just an adjunct management task. It is a vital step in the transition from a reactive culture, one in which the organization reacts to an event, or to a proactive culture, in which the organization actively seeks to address systemic safety issues before they result in an active failure. The fundamental purpose of a risk-management system is the early identification of potential problems. The risk-management system enhances the way management safety decisions are made. The risk-management process identifies the following six steps:

1. Establish the Context. This is the most significant step of the risk process. It defines the scope and definition of the task or activity to be undertaken, defines the acceptable level of risk, and determines the necessary level of risk-management planning.
2. Identify the Risk. Identification of what could go wrong and how it can happen is examined, hazards are also identified and reviewed, and the source of risk or the potential causal factors are also identified.
3. Analyze the Risk. Determines the likelihood and consequence of risk to calculate and quantify the level of risk. A good tool for this process is the reporting system for the information-gathering technique. Determining the frequency and consequence of past

occurrences can help to establish a baseline for your risk matrix. Each organization will have to determine its definition of severity according to its individual risk aversion.

4. Evaluate the Risks. Determines whether the risk is acceptable or whether the risk requires prioritization and treatment. Risks are ranked as part of the risk analysis and evaluation step.
5. Treat the Risks. Adopts appropriate risk strategies to reduce the likelihood or consequence of the identified risk. These could range from establishing new policies and procedures, reworking a task, making a change in training, or giving up a particular mission or job profile.
6. Monitor and Review. This is a required step at all stages of the risk process. Constant monitoring is necessary to determine if the context has changed and if the treatments remain effective. In the event the context changes, a reassessment is required.

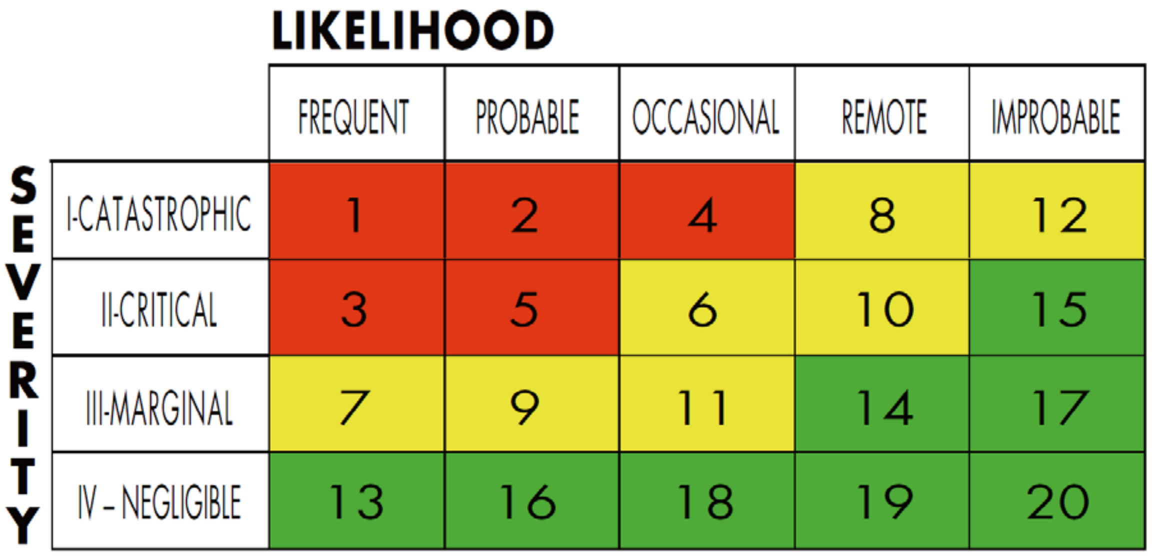
Risk Assessment

Risk assessment is a decision step, based on combined severity and likelihood. Ask if the risk is acceptable. The risk assessment may be concluded when potential severity is low or if the likelihood is low or well controlled.

Risk Matrix

The risk-assessment matrix is a useful tool to identify the level of risk and the levels of management approval required for any risk-management plan. There are various forms of this matrix, but they all have a common objective to define the potential consequences/ severity of the hazard versus the probability or likelihood of the hazard.

To use the risk-assessment matrix effectively, it is important that everyone has the same understanding of the terminology used for probability and severity. For this reason, definitions for each level of these components should be provided. Figure A-2 shows a risk matrix used by many aviation organizations.



		LIKELIHOOD				
		FREQUENT	PROBABLE	OCCASIONAL	REMOTE	IMPROBABLE
SEVERITY	I-CATASTROPHIC	1	2	4	8	12
	II-CRITICAL	3	5	6	10	15
	III-MARGINAL	7	9	11	14	17
	IV - NEGLIGIBLE	13	16	18	19	20

Figure A-2. Risk matrix, FAA

Risk Control

Often, risk mitigation will require new processes, new equipment, or changes to existing ones. One should look at the system with the proposed control in place to see if the level of risk is now acceptable.

One should stay in this design loop until it is determined that the proposed operation or change is not mitigated to allow operations within acceptable levels of risk.

Safety Assurance

Policies, process measures, assessments, and controls are in place. The organization must incorporate regular data collection, analysis, assessment, and management review to assure safety goals are being achieved. Solid change management processes must be in place to assure the system is able to adapt.

The ongoing monitoring of all systems and the application of corrective actions are functions of the quality-assurance system. Continuous improvement can occur only when the organization displays constant vigilance regarding the effectiveness of its technical operations and its corrective actions. Without ongoing monitoring of corrective actions, there is no way of telling whether the problem has been corrected and the safety objective met. Similarly, there is no way of measuring if a system is fulfilling its purpose with maximum efficiency. Evaluation of the safety program includes external assessments by professional or peer organizations. Safety oversight is provided in part by some of the elements of the SMS, such as occurrence reporting and investigation. However, safety assurance and oversight programs proactively seek out potential hazards based on available data and the evaluation of the organization's safety program. This can best be accomplished by:

- Conducting internal assessments of operational processes at regularly scheduled intervals.

- Using checklists tailored to the organization’s operations when conducting safety evaluations.
- Assessing the activities of contractors if their services may affect the safety of the operation.
- Having assessment of evaluator’s processes conducted by an independent source.
- Documenting results and corrective actions.
- Documenting positive observations.
- Categorizing findings to assist in prioritizing corrective actions.
- Sharing the results and corrective actions with all personnel.
- Using available technology, such as health usage monitoring systems (HUMS), to supplement quality and maintenance programs and flight data monitoring to evaluate aircrew operations.
- Facilitating safety committee meetings.
- Advising the CEO (accountable executive) on safety issues.
- Causing incidents to be investigated and reviewed, making recommendations, and providing feedback to the organization.
- Conducting periodic assessment of flight operations.
- Providing safety insight to the organization’s management.

Monitoring by audit forms a key element of this activity and should include both a quantitative and qualitative assessment. The results of all safety-performance monitoring should be documented and used as feedback to improve the system.

It is widely acknowledged that accident rates are not an effective measurement of safety. They are purely reactive and are only effective when the accident rates are high enough. Furthermore, relying on accident rates as a safety-performance measurement can create a false impression, an assumption that zero accidents indicate the organization is safe. A more effective way to measure safety might be to address the individual areas of concern. For example, an assessment of the improvements made to work procedures might be far more effective than measuring accident rates.

Interfaces in Safety Risk Management (SRM) and Safety Assurance

Safety risk management (SRM) and safety assurance are the key functional processes of the SMS. They are also highly interactive. The flowchart in figure A-3 may be useful to help visualize these interactions. The interface element concerns the input-output relationships between the activities in the processes. This is especially important if interfaces between processes involve interactions between different departments and contractors. Assessments of these relationships should place special attention to flow of authority, responsibility, communication, and procedures and documentation.

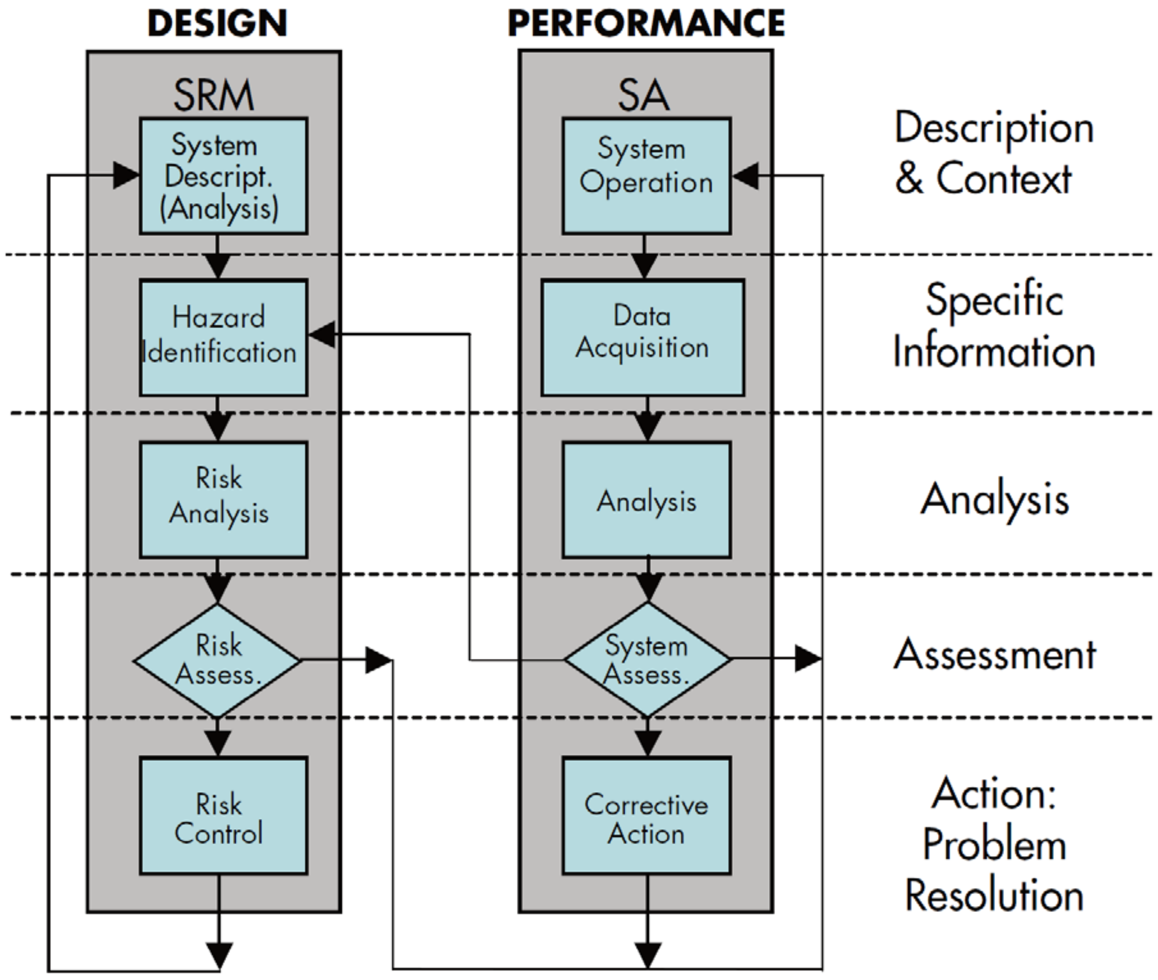


Figure A-3. Interaction between SRM and safety assurance

Safety Promotion and Safety Culture

The organization must continually promote, train and communicate safety as a core value with practices that support a sound safety culture.

An organization's safety culture influences the values, beliefs, and behaviors shared with other members of the various social groups. Culture serves to bind people together as members of groups and to provide clues as to how they behave in both normal and unusual situations. Some people see culture as the "collective programming of the mind." Culture is the complex, social dynamic that sets the rules of the game or the framework for all interpersonal interactions. It is the sum total of the way people work. Culture provides a context in which things happen. For safety management, understanding the culture is an important determinant of human performance and its limitations. The ultimate responsibility for safety rests with the management of the organization. Safety culture is affected by such factors as:

- Management's actions and priorities.
- Policy and procedure.
- Supervisory practices.
- Safety planning and goals.
- Actions in response to unsafe behaviors.
- Employee training and motivation.
- Employee involvement or buy-in.

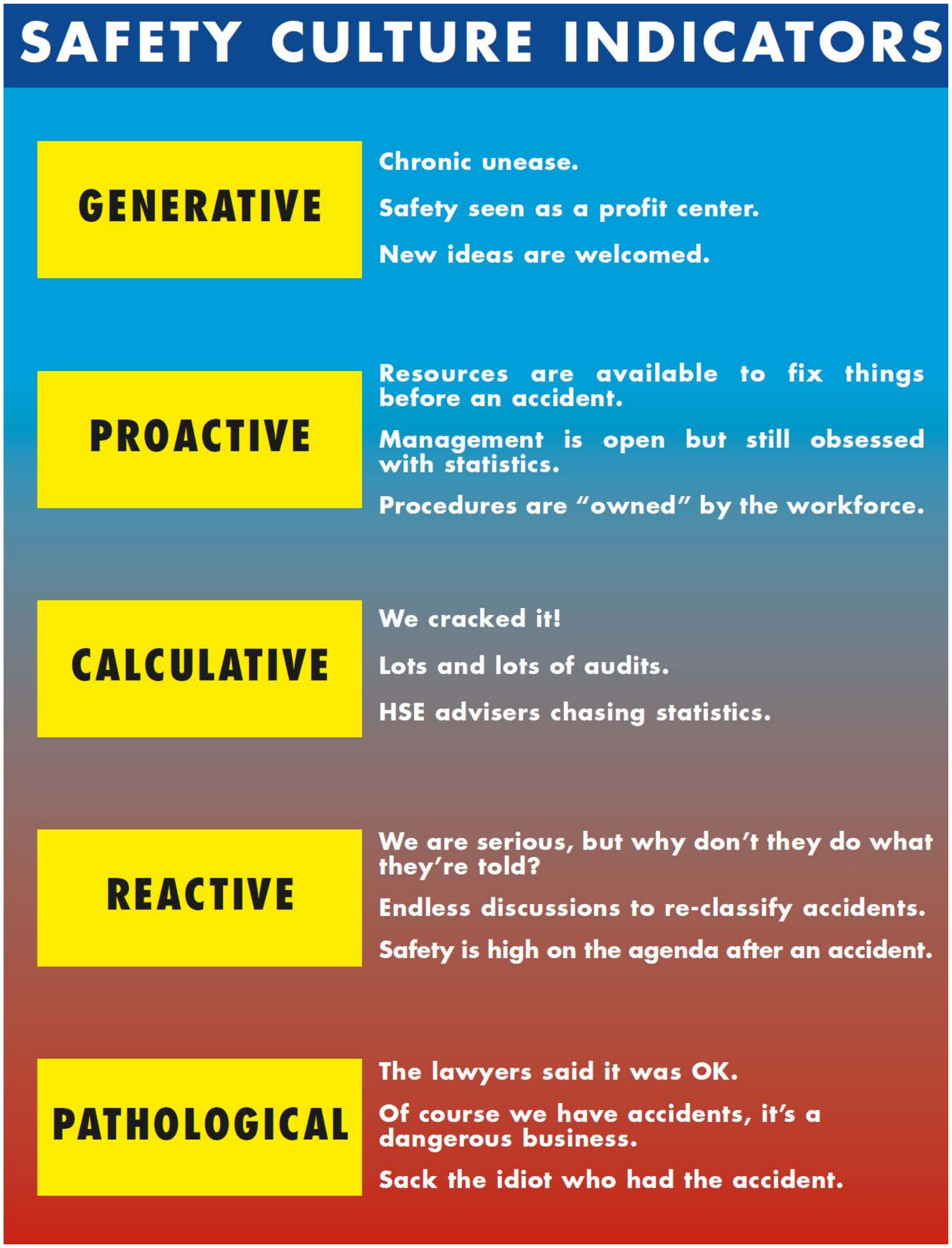
An organizational culture recognizes and identifies the behavior and values of particular organizations. Generally, personnel in the aviation industry enjoy a sense of belonging. They are influenced in their day-to- day behavior by the values of their organization. Does the organization, for example, recognize merit, promote individual initiative, encourage risk taking, tolerate breeches of SOPs, and promote two-way communication? The organization is a major determinant of employee behavior.

Positive Safety Culture

A positive safety culture is generated from the top down. It relies on a high degree of trust and respect between workers and management. Workers must believe that they will be supported in any decisions made in the interest of safety. They must also understand that intentional breaches of safety that jeopardize operations will not be tolerated. A positive safety culture is essential for the effective operation of an SMS. However, the culture of an organization is also shaped by the existence of a formal SMS. Therefore, an organization should not wait until it has achieved an ideal safety culture before introducing an SMS. The culture will develop as exposure and experience with safety management increases. Figure A-4 shows common attributes of a positive safety culture.

Indications of Positive Safety Culture:

- Senior management places strong emphasis on safety as part of the strategy of controlling risks and minimizing losses.
- Decision-makers and operational personnel hold realistic views of the short- and long-term hazards involved in the organization's activities.
- Management fosters a climate in which there is a positive attitude toward criticisms, comments, and feedback from lower levels of the organization on safety matters.
- Management does not use their influence to force their views on subordinates.
- Management implements measures to minimize the consequences of identified safety deficiencies.



SAFETY CULTURE INDICATORS

GENERATIVE	Chronic unease. Safety seen as a profit center. New ideas are welcomed.
PROACTIVE	Resources are available to fix things before an accident. Management is open but still obsessed with statistics. Procedures are "owned" by the workforce.
CALCULATIVE	We cracked it! Lots and lots of audits. HSE advisers chasing statistics.
REACTIVE	We are serious, but why don't they do what they're told? Endless discussions to re-classify accidents. Safety is high on the agenda after an accident.
PATHOLOGICAL	The lawyers said it was OK. Of course we have accidents, it's a dangerous business. Sack the idiot who had the accident.

Figure A-4. Common attributes of a positive safety culture

Safety must not only be recognized but promoted by the senior management team as the organization’s primary core value. Procedures, practices, training, and the allocation of resources clearly demonstrate management’s commitment to safety.

The key elements of promoting safety within any organization are:

- Safety Culture–Support the expansion of a positive safety culture throughout the organization by:

    - Widely distributing and visibly posting organizational safety policy and mission statements signed by senior management.
    - Clearly communicating safety responsibilities for all personnel.
    - Visibly demonstrating commitment to safety through everyday actions.
    - Implementing a “just culture” process that ensures fairness and open reporting in dealing with human error.

- Safety Education.
- Widely communicated status on safety performance related to goals and targets.
- Communication of all identified safety hazards.
- Overview of recent accidents and incidents.
- Communication of lessons learned that promote improvement in an SMS.
- Safety training.
- Initial new-employee safety training.
- Recurrent safety training for all employees.
- Document, review, and update training requirements.
- Define competency requirements for individuals in key positions.
- Introduce and review safety policies.
- Review of safety-reporting processes.
- Safety communication.
- Communicate the realized benefits of an SMS to all employees.
- Implement a safety feedback system with appropriate levels of confidentiality that promote participation by all personnel in the identification of hazards.
- Communicate safety information with employees through:
    - Safety newsletters
    - Bulletin board postings
    - Safety investigation reports
    - Internet website

Black’s Law Dictionary, a popular legal reference, defines assurance as something that gives confidence. Therefore, SA might be defined as activities designed to gain assurance.

The management of change exists within the SA function. It applies the activities of safety assurance and internal evaluation to ensure that risk controls, once designed, continue to conform to their expectations and that they continue to be effective in maintaining risk within acceptable levels. These assurance and evaluation functions also provide a basis for continuous improvement.

How does an SMS differ from traditional approaches? Implementing an SMS does not involve the regulator imposing an additional layer of oversight on the aviation industry. The traditional flight-safety approach depended on a flight-safety officer or a department in a large organization, independent from operations management, but reporting to the chief executive of the organization. The safety officer had no authority to make changes that would enhance safety. The safety officer depended on his/her ability to persuade management to act. An SMS holds managers accountable for safety-related performance.

Change is the catalyst for hazard identification

Change is the catalyst for performing the hazard-identification and risk-management process. Some examples of change include but are not limited to:

- Organizational structure.
- Acquisition of equipment.
- Fleet make-up.
- Mission content or type.
- Personnel management.
- Regulations.
- Competition.
- Customer base.
- Security.
- Financial status.

Routines and habits are parts of "the way things have always been done." This can be counterproductive to actually affecting change. People get used to doing things a certain way. Change is often viewed negatively because it is something different and not part of the normal routine. When change becomes necessary, it is vital to involve affected personnel in the process to gain buy-in, acceptance, and ownership in the management of change process. All involved in the process must be aware of what needs to change and why. It is important to engage and motivate staff to create an atmosphere of understanding what change can bring to move through the entire process of change.

All organizations, regardless of size, are involved in continual change. The traditional model used in dealing with change is based on a specific desired outcome through solving very specific and limited tasks/jobs. People evolve into a process of management of change focused on continuous improvement in the process (the way they do things every day). Unless properly managed, changes in organizational structure, personnel, documentation, processes, or procedures can result in the inadvertent introduction of hazards, resulting in increased risk. Good organizations continually seek to find ways to improve processes, recognizing that changes need to be properly and effectively managed. The organization can minimize the likelihood of introducing risk associated with change by:

- Analyzing changes in operational procedures or processes to identify required changes in training, documentation, or equipment.
- Analyzing changes in location of equipment or operating conditions for potential hazards.

- Ensuring all maintenance and operations manuals are kept up-to-date with the most current changes.
- Having a process to ensure all personnel are aware and understand changes in requirements, procedures, and applicable maintenance and operations manuals.
- Defining the level of management to approve a change.

Change Process

Because the transformational process is difficult by nature, there has to be a strong motivation to stick with the effort. This motivation must exist throughout the team, or the person in a small organization, assigned to achieve the goal of the project. A sense of urgency must exist at the beginning of the process at the top of the organization. In a small organization, the champion may be a one-person band. The SMS champion must convince top management, to the greatest extent possible, what the organization must do and who must do it. The champion should remind leaders that the ICAO has mandated an SMS in its member states, and each of those states is in the process of accomplishing the mandate. No matter what their initial level of support for an SMS, people understand the wisdom of quick adoption of programs they will eventually have to do anyway. In a small organization, the champion may be the only member of the implementation team. In larger organizations, it is important the champion be empowered to select members of the SMS implementation team from across all departments. The members of the implementation team do not have to be top-level managers. In fact there is much to be said for team members not being part of management. Although team members should not be the executive VP's, they should have a high degree of respect within the organization because they are going to be leading their coworkers into a new way of doing business. It is wise when making initial changes to go for the short-term wins. Short-term wins create confidence in the process and help motivate others to join in the change effort in this and other areas. The choice for the first area of implementation is important and has a meaningful impact on safety. Even more important, however, is that it be achievable. It is valuable to consider the entire operation and choose an area that represents a significant portion of the organization's risk exposure (i.e., flight operations). The team is going for a big win, so be aware that going after too big of a program can be risky.

MOC Phases

The management of change (MOC) process has four basic phases: screening, review, approval, and implementation. Both the effect of change and the effect of implementing change are considered. The systematic approach to managing and monitoring organizational change is part of the risk-management process. Safety issues associated with change are identified, and standards associated with change are maintained during the change process. Procedures for managing change include:

- Risk assessment.
- Identifying the goals, objectives, and nature of the proposed change.
- Identifying operational procedures.
- Analyzing changes in location, equipment, or operating conditions.
- Posting current changes in maintenance and operator manuals.
- Making all personnel aware of and understanding changes.
- Identifying the level of management with authority to approve changes.

- Reviewing, evaluating, and recording potential safety hazards from the change or its implementation.
- Approval of the agreed on change and the implementation procedure(s) There are methods for managing the introduction of new technology.

All personnel should be consulted when changes to the work environment, process, or practices could have health or safety implications. Changes to resource levels and competencies associated with risks are assessed as part of the change control procedure. Regardless of the magnitude of change, there must always be consideration for safety, the associated risks, and the management of change principles.

Change can be successful only if personnel are engaged, involved, and participate in the process management. Management of change provides a structured framework for managing all aspects of the change. How change is introduced dramatically impacts the implementation and effectiveness of the outcome. Procedures are established and maintained to manage change with a specific focus on safety and risk. Throughout the process, it is important that all personnel involved have an accurate understanding of what must be changed and whyit must be changed. It is imperative that management personnel provide direction, guidance, and in-depth communication. The structure and responsibilities associated with change are defined prior to introducing any change. It is important to recognize the complexity of change prior to, during, and subsequent to the change itself. Anticipate unintended consequences and the necessity to redirect the process if change fails. Change can fail for many reasons, some of which are as follows:

- Lack of top down support
- Loss of control
- Insufficient resources
- Commitment changes
- Poor communication of the process
- Lack of clarity and consistency
- Lack of understanding
- Insufficient risk analysis
- Timelines that are too aggressive

The systematic approach to managing and monitoring organizational change is part of the risk-management process. Safety issues associated with change are identified, and standards associated with change are maintained throughout management of the change process.

Once the need for change has been identified, a structured process is followed for change to be appropriately managed. Procedures for managing change:

- Change recognition based on differing elements.
- Planned—Introduction of new product (aircraft-technology).
- Unplanned—Response to outside influences, such as regulatory or market factors
- Description.
- Create a vision of the change.
- Enable affected personnel to be aware and to become involved in later stages.
- Classification.

- Determine the magnitude of the proposed change.
- Identify route to be followed for change.
- Identify objectives and constraints.
- Detail the objectives of the change.
- Identify both internal and external constraints impacting change.
- Evaluate the necessity for redefining organization standards to fit the change.
- Initial design.
- Develop potential plans for implementation of the change.
- Generation and evaluation of options and differing paths to bring about the change.
- Detail design.
- Initial processes produce a detailed procedure for the change.
- Detailed process will justify review and support approval for implementation.
- Implementation.
- According to detailed change plan.
- Plan defines the monitoring and processes to be implemented.
- Feedback and followup.
- Widely communicate the change process to personnel.
- Periodic review of management of change for effectiveness.

A change-management process should identify changes within the organization, which may affect established processes, procedures, products, and services. Prior to implementing changes, a change-management process should describe the arrangements to ensure safety performance. The result of this process is the reduction in the safety risks resulting from changes in the provision of products or services by the organization. Change management should consider the criticality of the system and activities, the stability of the system and operational environment, and past performance of the system.

The following table identifies the FAA's performance objectives and design expectations for the Management of Change. The design expectations consist of inputs, management responsibilities, procedures, outputs and measures, and controls.

Table A-1. FAA's performance objectives and design expectations for the Management of Change

Management of Change
Performance Objective
The organization's management will identify and determine acceptable safety risk for changes within the organization that may affect established processes and services by new system design, changes to existing system designs, new operations/procedures, or modified operations/procedures.
Design Expectations
Input
Inputs (interfaces) for this process will be obtained from proposed changes to systems, processes, procedures, or organizational structures. *Reference: SMS Framework 1.5 b, (1) (f)* (I)
Management Responsibility
The organization will clearly identify who is responsible for the quality of the Management of Change Process. Procedures will also define who is responsible for accomplishing the process. *Reference: SMS Framework 1.2 b, (3)* (R/A)
Procedure
Does the organization ensure it does not implement any of the following until the level of safety risk of each identified hazard is determined to be acceptable for:
New system designs? *Reference: SMS Framework 3.2 b, (1) (a)* (P)
Changes to existing system designs? *Reference: SMS Framework 3.2 b, (1) (b)* (P)
New operations or procedures? *Reference: SMS Framework 3.2 b, (1) (c)* (P)
Modifications to existing operations or procedures? *Reference: SMS Framework 3.2 b, (1) (d)* (P)
Outputs and Measures
The organization will: (1) ensure that this process is interfaced with the SRM process (System Description and Task Analysis 2.1.1), and (2) periodically measure performance objectives and design expectations of the Management of Change Process. *Reference: (1) SMS Framework 1.5 b, (1) (f): (2) SMS Framework note at 3.1.3 & 1.0 b, (2) (c) and (3) (c); 3.1.3 b, (1)* (PM/I)
Controls
The organization will ensure that: (1) procedures are followed for safety-related operations and activities, and (2) they periodically review supervisory and operational controls to ensure the effectiveness of the Management of Change Process. *Reference: (1) SMS Framework: 1.0 b, (4) (f): (2) SMS Framework 1.1 b, (2) k); 3.1.3 b, (1); 3.3.2, b, (1) & (2)* (C)
Bottom Line Assessment
Has the organization's management assessed risk for changes within the organization that may affect established processes and services by new system designs, changes to existing system designs, new operations/procedures or modified operations/procedures?

All SMS information is consistent with the information and guidance contained in other documents including:

- ICAO Doc 9859 Safety Management Manual
- FAA SMS Framework, SMS Assurance Guide and SMS Implementation Guide, as revised (these documents are the nucleus of the FAA Advisory Circular (AC) 120-92A.
- FAA 14 CFR Part 5.
- FAA SMS Voluntary Program
- FAA SMS Framework & Assurance Guide – Rev. 2
- Transport Canada Safety Management Manual TP 13739