
Considerations in Assuring Safety of Increasingly Autonomous Systems

Final Technical Report for NASA Project
Assurance Reasoning for Increasingly Autonomous Systems
(*ARIAS*)

NASA Contract: NNL16AA06B Task Order: NNL16AA96T

May 2018

Erin Alves, Devesh Bhatt, Brendan Hall, Kevin Driscoll, Anitha Murugesan
Aerospace Advanced Technology Labs
Honeywell International, Inc.

John Rushby
SRI International

Contents

1	Introduction	1
1.1	Context and Scope	1
1.2	Report Organization	2
2	Challenges in Safety Assurance of IA Systems	4
2.1	Potential Failures and Safety concerns in Autonomous Systems	4
2.2	Levels of Autonomy	7
2.3	Challenges in Assuring Safety of IA Systems in Reduced Crew Operations	10
2.3.1	The Challenge of Integration and Autonomy	10
2.3.2	The Challenge of CRM	11
2.3.3	The Challenge of Never Give up	13
2.3.4	The Challenge of Unconventional Implementations	14
2.4	Shortfalls of Traditional Assurance Approach for Future IA Systems	15
3	Safety Assurance Approach for IA Systems	18
3.1	Safety Assurance Context and Aspects	18
3.2	Rationale for Assurance Case and Property-Based Assurance	20
3.2.1	Property-Based Assurance.	21
3.3	Framework for IA Assurance Case	27
4	Safety Relevant Requirements in IA Systems	30
4.1	Operational Scenarios	30
4.1.1	Critical Events	32
4.2	Cognitive Work Analysis	32
4.3	Stakeholder-Level Requirements for an IA System in CRM Context	36
4.3.1	Technical Assumptions	36
4.4	Property-Based Specification of IA System-Level Requirements	37
4.4.1	Property-Based Requirements	37
4.5	Requirements Categorization	39
4.5.1	Instrument Monitoring	39
4.5.2	System Monitoring	40
4.5.3	Environment Monitoring	42
4.5.4	Flight Controls and Navigation	43
4.5.5	Fly manually (Direct Control Input)	45
4.5.6	Crew Communication	46
4.5.7	Communicate with ATC	48
4.5.8	Check Reference Material	49
4.5.9	Calculations	49
4.5.10	Troubleshooting and Checklists Actions	50
4.5.11	Ground Maneuvering	51

4.6	Addressing Communication and Trust	52
5	Safety Analysis Techniques	56
5.1	Overview of STPA	57
5.2	Tooling	58
5.3	Landing Gear Deployment - Case Example	59
5.4	Discussion	67
6	System Architecture Considerations	69
6.1	Establishing the System Context	71
6.2	Guided and Principled Architectural Refinement	73
6.3	System Architecture and Pervasive Monitoring	74
6.4	Overview of the Approach for System Architecture when IA Systems are Used	77
6.4.1	System Dependability Requirements in the Broad Context	77
6.5	Impact of System Concepts and Design Choices on Architecture	78
6.5.1	Human-IA System Interactions and Control Authority Considerations	78
6.5.2	Situational Awareness: Additional Sensors and Processing Considerations	80
6.5.3	IA System Integration into Avionics	81
6.5.4	Distributed Vs Centralized processing	85
6.5.5	Resource/Failure Analyses and Dependability Considerations	85
6.6	Autonomy Architecture for Assurance	87
7	V&V Issues and Approaches	91
7.1	Issues in V&V of Machine Learning Techniques	92
7.1.1	V&V Issues in Unsupervised Learning	92
7.1.2	V&V Issues in Supervised Learning	94
7.1.3	V&V Issues in Reinforcement Learning	95
7.2	V&V Approaches for Specific Machine Learning Implementations	96
7.2.1	V&V Approaches in Artificial Neural Networks	96
7.2.2	Safe Reinforcement Learning	99
7.3	Compositional Verification Challenges	101
7.4	Challenges in Testing of IA Systems	102
7.5	Challenges and Approaches for Simulation of Human-IA Interactions	103
7.5.1	Initial and Periodic Knowledge Checks	104
7.5.2	Understanding the Human Crewmate	106
7.5.3	Crew Resource Management	108
7.5.4	Roles, Responsibilities and Task Management	109
7.5.5	Fault Management	112
7.5.6	Impact of Human Automation Teaming (HAT) on V&V Methods	113
8	Future Work Directions for Safety Assurance of IA Systems	115
8.1	Top-Down Systems Approach for Safety Assurance of IA systems	115

8.2	Pervasive Monitoring as a Major Component of overall System Assurance	117
8.3	Safety Assurance Approach for Human-IA Interaction	117
8.4	Automated Reasoning to Assist in Architecture and Safety Analyses	119
8.5	Autonomous Ethics	120
9	Conclusions	121
	References	122
A	An Examination of Agent Safety Logic	133
B	Requirements of IA Systems in Reduced Crew Operations	154
B.1	Information Requirements	154
B.2	Functional Requirements	156
B.3	Communications Requirements	158

List of Figures

1	Sheridan levels of automation	8
2	SAE levels of automation	9
3	Context for Safety Assurance	19
4	A Structured Argument in Free and Simple Form	28
5	Phases Of A Flight (<i>Image courtesy of Federal Aviation Administration 4.1 Taxi</i>)	33
6	Visual Meteorological Conditions	43
7	Explainable AI (XAI) Concept	53
8	Accident and Hazard List view	61
9	Hierarchical Control Structure - Operational Process Control Level	62
10	Hierarchical Control Structure - Management Level	63
11	Unsafe Control Actions	65
12	Causal Factors for Unsafe Control Actions	66
13	Orthogonal Aspects of System Design (Architecture Instantiation)	71
14	Architecture Refinement using STPA	72
15	Conventional IA Monitor Architecture	75
16	Pervasive Monitor Architecture	75
17	ALIAS Inadvertent Hazardous Action	82
18	Centralized “Porcupine” Topology	84
19	Remote Agents Topology	84
20	Unsupervised Learning for Clustering and Anomaly Detection	93
21	Envelope with Modes	94
22	Artificial Neurons and Artificial Neural Net	97
23	Authority and Autonomy in Cooperative Human-IA interaction	111
24	Specification of Ahrenbach and Goodloe’s Basic Theory in PVS	134
25	Specification of Ahrenbach and Goodloe’s Axioms in PVS	135

List of Tables

1	Cognitive Work Analysis Quick Look	34
2	Cognitive Work Analysis Quick Look. Contd.	35
3	Information Updating	154

Acronyms and Abbreviations

AGL	Above Ground Level
AI	Artificial Intelligence
ALIAS	Aircrew Labor In-Cockpit Automation System
AMDP	Abstract Markov Decision Process
AN	Artificial Neurons
ANN	Artificial Neural Network
ARIAS	Assurance Reasoning for Increasingly Autonomous Systems
ARL	Assured Reinforcement Learning
ASRS	Aviation Safety Reporting System
AT	Airline Transport
ATC	Air Traffic Control
ATIS	Airport Tower Information Service
BRGA	Business, Regional and General Aviation
CAE	Claims-Argument-Evidence
COTS	Commercial Off-The-Shelf
CRM	Crew Resource Management
CTM	Crew task management
CWA	Cognitive Work Analysis
DAL	Design Assurance Level
DARPA	Defense Advanced Research Projects Agency
EA	Enterprise Architect
EFB	Electronic Flight Bag
EGPWS	Enhanced Ground Proximity Warning Systems
FAA	Federal Aviation Administration
FHA	Functional hazard analysis
FL	Flight Level
FMEA	Failure Modes and Effects Analysis
FMS	Flight Management System
FO	First Officer
FPM	Flight Path Management
FTA	Fault Tree Analysis
GPWS	Ground Proximity Warning Systems
GSN	Goal Structuring Notation
HITL	Humans in the Loop
IA	Increasingly autonomous
IAS	Intelligent Autopilot System

IFR	Instrument Flight Rules
IMC	Instrument Meteorological Conditions
LOA	Levels of Automation
MCP	Mode Control Panel
MDP	Markov Decision Process
ML	Machine Learning
MSL	Mean Sea Level
Nav DB	Navigation Database
NGU	Never Give Up
NL	Natural Language
NLD	Natural Language Deductivism
NOTAM	Notice to Airmen
NTSB	National Transportation Safety Board
PBR	Property-Based Requirements
PF	Pilot Flying
PM	Pilot Monitoring
PVS	Prototype Verification System
RA	Requirements Analysis
RL	Reinforcement Learning
RNAV	Area Navigation
ROAAS	Runway Overrun Awareness and Alerting Systems
SA	Situation Awareness
SAHRA	STPA based Hazard and Risk Analysis
SEU	Single Event Upset
SMM	Shared Mental Model
SMT	Satisfiability Modulo Theory
SOP	Standard Operating Procedures
SSA	System safety assessment
STAMP	System-Theoretic Accident Model and Process
STPA	Systems Theoretic Process Analysis
SysML	Systems Modeling Language
TOGA	Take-Off/Go Around
UCA	Unsafe Control Actions
UML	Universal Modeling Language
V&V	Verification and Validation
VFR	Visual Flight Rules
VMC	Visual Meteorological Conditions
HAT	Human-Automation Teaming
NLP	Natural Language Processing
ToM	Theory of Mind

1 Introduction

Recent technological advances have accelerated the development and application of increasingly autonomous (IA) systems in civil and military aviation [34]. IA systems can provide automation of complex mission tasks—ranging across reduced crew operations, air-traffic management, and unmanned, autonomous aircraft—with most applications calling for collaboration and teaming among humans and IA agents. IA systems are expected to provide benefits in terms of safety, reliability, efficiency, affordability, and previously unattainable mission capability. There is also a potential for improving safety by removal of human errors.

There are, however, several challenges in the safety assurance of these systems due to the highly adaptive and non-deterministic behavior of these systems, and vulnerabilities due to potential divergence of airplane state awareness between the IA system and humans. These systems must deal with external sensors and actuators, and they must respond in time commensurate with the activities of the system in its environment. One of the main challenges is that safety assurance, currently relying upon authority transfer from an autonomous function to a human to mitigate safety concerns, will need to address their mitigation by automation in a collaborative dynamic context.

These challenges have a fundamental, multidimensional impact on the safety assurance methods, system architecture, and V&V capabilities to be employed. The goal of this report is to identify relevant issues to be addressed in these areas, the potential gaps in the current safety assurance techniques, and critical questions that would need to be answered to assure safety of IA systems. We focus on a scenario of reduced crew operation when an IA system is employed which reduces, changes or eliminates a human's role in transition from two-pilot operations.

1.1 Context and Scope

This report is the product of work performed on the Assurance Reasoning for Increasingly Autonomous Systems (ARIAS) project, funded by NASA under Contract: NNL16AA06B and Task Order: NNL16AA96T. The ARIAS project objective is to perform the following tasks in the context of reduced crew operations:

1. Identify safety requirements introduced when the human role in systems operation is reduced, changed or eliminated, by an IA system.
2. Identify changes necessary in systems safety assurance to account for changes to human roles (e.g. task responsibilities and decision-making authority).
3. Identify changes necessary in system architectures to provide an equivalent, or better, level of safety for IA systems.
4. Identify new Verification and Validation (V&V) capabilities needed to assure IA systems.

In the first task, we applied the principles of crew resource management (CRM) between two human pilots to identify new tasks that automated systems will need to accomplish and interaction scenarios

with the human pilot. We explore the requirements in the above context at the stakeholder level and further at the system level—leading to a property-based requirements specification. We analyze the requirements from the perspective of the impact on design/safety assurance. Thus, the results of this task provide inputs and examples for the effort towards the other objectives of this project.

In the second task, we identified changes to safety assurance methods to account for changes in human and automation roles. In particular, we define a process for hazard analysis in such a scenario of IA system usage. We also determine the applicability of verifying functions to different degrees of assurance and support such assessment via a combination of simulation and model-checking within the structure provided by assurance cases.

In the third task, we identified changes to system architecture to provide an equivalent, or better, level of safety for IA systems and the challenges faced in the system architecture design and analysis. The goal is not to suggest a specific architecture for an IA system; rather, we enumerate the key architectural considerations influencing safety that one has to taken into account while designing an IA system. We focus on enumerating the impact, challenges, and potential approaches in various aspects of system architecture. These aspects relate to each other and also to the findings in the previous two technical report in this project.

In the fourth task, we identified capabilities needed to verify and validate IA systems, that predominately based on machine learning. We explore the state-of-the-art techniques that are being currently employed to design safe learning systems as well as those that attempt to verify such systems. Again, the goal is not to propose a specific technique, but to explore research and practical challenges that needs to be addressed to verify such systems with high-confidence.

1.2 Report Organization

This final report is a consolidation of the findings in the aforementioned tasks.

In Section 2, we provide an overview of the safety issues noted with IA systems in general applications, additional challenges in reduced crew operations, and shortfalls of current assurance methods.

In Section 3 we discuss several aspects of safety assurance that must be addressed in the broad context—subsection 3.1 provides a roadmap to the remaining sections of this report which elaborate on these aspects. We also present a framework for an assurance case in this section.

In Section 4, we provide examples and describe the challenges in specifying safety relevant requirements for IA Systems based on cognitive workload analysis.

In Section 5, we investigate the benefits/gaps in applying the state-of-the art techniques in safety analysis and methods to fill the gaps in fully understanding the risks and causal factors inherent in introducing an IA controller.

In Section 6, we elaborate on the impact of system concepts and design choices on the architecture and its considerations and methods, including techniques for safety analysis with architecture refinement

and use of automated reasoning to aid in the analysis of complex architectural concerns.

In Section 7, we discuss challenges and current approaches for verification and validation (V&V) of IA systems with high-confidence.

Finally, in Section 8, we explore future work directions and conclude in Section 9.

2 Challenges in Safety Assurance of IA Systems

Safety assurance of complex avionic systems is currently based upon prescriptive regulatory guidance for verification and validation (V&V) processes for traditional avionics and mechanical control systems with deterministic and limited automation functionality. Further, the current approach relies upon deterministic transfer of authority from an automated function to a human (such as pilots) to mitigate safety concerns.

Safety assurance of future IA systems must address the highly adaptive and nondeterministic behavior of these systems and vulnerabilities arising due to potential divergence of airplane state awareness between automation and humans—with mitigation of safety concerns by automation in a collaborative dynamic context. Furthermore, IA systems must provide highly ‘intelligent’ functions, dealing with a variety of external sensors and actuators, and they must respond in time commensurate with the activities of the system in its environment.

In this section, we first provide a summary of the safety concerns noted by practitioners in deploying autonomous systems in various domains such as autonomous vehicles and robotics. We then discuss some specific challenges faced in assuring safety of IA Systems in reduced crew operations context. Finally, we discuss the potential shortfalls of the traditional avionics assurance approach in meeting these challenges.

2.1 Potential Failures and Safety concerns in Autonomous Systems

In this subsection, we summarize the safety concerns noted by researchers and practitioners in developing autonomous systems in various domains such as autonomous vehicles and robotics.

Artificial Intelligence (AI) and Machine Learning (ML) refer to a class of techniques that are being increasingly used in autonomous systems that attempt to mimic the adaptive and smart problem-solving capabilities of humans. Such systems promise a smarter and safer world—where self-driving vehicles can reduce the number of road accidents, medical robots perform intricate surgeries, and “digital” pilots participate in crew flight-operations. Despite the overwhelming benefit in the future promised by the AI technologies, concerns of *technical debt* [119] arise in that it is easy to incur massive ongoing maintenance costs at the system level when applying these techniques, especially for safety-critical ones. Machine learning modules have all the basic safety issues as normal ones, but also have a broader impact to the system that can create *hidden debt*. Machine learning modules may often be treated as black boxes, resulting in difficulties in formal verification and diagnosis. It is difficult to enforce strict abstraction boundaries for ever-evolving machine learning modules by setting up formal requirements to specify intended behaviors. Indeed, arguably the most important reason for deploying machine learning modules is that the desired behavior cannot be effectively implemented in explainable logic without dependency on external data. Therefore, things like wrongly interpreted formal objective function (reward) and changes in the external world may make modules alter behavior in unintended and unpredictable ways.

The notion of safety is domain specific; there are some loose notions of safety for machine-learning based autonomous systems. In [98, 99], and numerous references therein, safety is defined to be the reduction or minimization of *risk* and *epistemic uncertainty* associated with unintended hazardous outcomes. Roughly speaking, risk is expected value of the cost incurred due to hazardous outcomes, while epistemic uncertainty results from the lack of knowledge that could be obtained in principle, but may be practically intractable to gather or undetermined. Amodei et al. [11] gave an overview of the various safety concerns and hazardous situations that are likely associated with AI systems and categorized safety problems depending upon where things went wrong, which we summarize below. First, when the formal objective function is specified, it may be oversimplified, or the trade-off is overlooked, due to lack of thorough consideration by the designer, such that maximizing that objective function leads to hazardous results, even assuming perfect learning and infinite data. Such design failures can be broadly attributed to two mechanisms as *negative side effects* and *reward hacking*:

- **Negative Side Effects:** The objective function focuses on accomplishing some specific task, but ignores other aspects of the (potential very large) environment, and thus implicitly expresses indifference over environmental variables that might actually be hazardous to change. The challenge here is that the *side effects* are highly idiosyncratic to each individual task due to the environmental change, therefore it is often impossible to hard-code all the constraints on everything the system should not disturb without run-time human intervene. A successful approach might be “smart” enough to be transferable across tasks, such intelligence requires the formalization of “change to the environment” or a learning-based environmental-impact regularizer.
- **Reward Hacking:** The objective function admits some “easy” solution that can be “gamed” to pervert the design intent. For example, an AI player of a video game may exploit a bug in the game to achieve high scores—since from the agent’s point of view, this is simply how the environment works even though a human may not perceive this as a “bug”. Pursuit of these *reward hacks* is an attempt to explore the difference between formal “rewards” and designer’s informal but true intent. There are several ways in which the problem can occur. For example, often only the partially observed system/environmental states are available to the agent, so the rewards are forced to represent imperfect measure. Further, the correlation between rewards and intent breaks because the objective function has a self-amplifying feedback loop component.

Second, the objective function itself may be correct or at least subject to proper evaluation (for example explicitly consulting a human on a given situation), but it is too expensive to do so frequently, leading to possible hazardous behavior by bad extrapolations from limited samples.

- **Limited Oversight Budget:** When an autonomous system performs some complex task, the complex and expensive objective function is often broken down into cheaper approximations which can be efficiently trained, but don’t perfectly track the task intent. This divergence exacerbates problems like negative side effect and reward hacking. The challenge is to search for more efficient ways to exploit limited oversight budget, for example by combining limited calls

to the true objective function with frequent calls to cheaper proxies.

Third, hazards can occur due to making decisions from insufficient or poorly curated training data or an insufficiently expressive model.

- **Safe Exploration:** When an autonomous system engages in exploration, it takes actions whose consequences are yet to understand. In real world the consequences could cause serious safety issues, for example leading a vehicle into crash. Common exploration policies such as epsilon-greedy or R-max [26] in reinforcement learning explore by choosing an action at random or viewing unexplored actions optimistically, and thus make no attempt to avoid these dangerous situation. In practice, reinforcement learning based system can often avoid these issues by simply hard-coding an avoidance of catastrophic behaviors, since it usually takes only a tiny bit of prior knowledge about certain situations to determine safety boundaries. But as systems become more autonomous and act in more complex domains, it may become harder and harder to anticipate every possible catastrophic failure.
- **Distributional Shift:** In general, when the testing distribution differs from the training distribution, machine learning systems may not only exhibit poor performance, but also wrongly assume their performance is good. The real world data will almost certainly differ from the training data, violating the common independent and identically distributed (i.i.d.) assumption in machine learning. Moreover, conventional neural networks provide no principled measure of how certain they are, or how well-trained they are to make a given prediction. Having a way to detect such failures—and ultimately having a statistical assurance case—is critical to building safe and predictable autonomous systems.

Additionally, the stability issue of a deep neural network (DNN) is addressed in [71]. DNN systems have achieved impressive experimental results in image classification, matching the cognitive ability of humans in complex tasks with thousands of classes. Many applications are envisaged, including their use as perception modules and end-to-end controllers for autonomous vehicles. Unfortunately, it has been observed that DNN modules, including highly trained and optimized networks for vision tasks, are unstable with respect to so called *adversary perturbation*. Su et al. [131] have shown that even one pixel attack (crafted using algorithms) can fool deep neural networks on 73.8% of the set of attack images with an average confidence of 98.7%. This renders the networks extremely vulnerable under hostile attacks.

- **Adversary Perturbation:** It is defined as minimal changes to the input image, often imperceptible to the human eye, that cause the network to misclassify the image. Examples include not only artificially generated random perturbations, but also (more worryingly) modifications of camera images that corresponding to resizing, cropping or change in light conditions. They can be devised independent to the training set and are transferable, in the sense that an example misclassified by one network is also misclassified by a network with a different architecture, even trained on different data.

Approaches towards safe learning developed over recent years are still largely theoretical, suffer from scalability issues, have difficulty in expressing non-trivial safety properties, or are unable to offer firm guarantee that their solutions will satisfy the safety requirements, and instead only offer “safer” solutions than the ones learned by traditional approaches. Mason et al. [89, 90] proposed further study towards *assured reinforcement learning* (ARL) to address current limitations (see details in Section 7.2.2), by providing the reinforcement learning system with safety policies abstracted from domain expert inputs and verified using quantitative verification technique of stochastic models. However, its assumption about full access to precise information is not likely to be satisfied by practical complex applications. Garcia et al. [57] provided a comprehensive survey on safe reinforcement learning which segmented safe solutions into two fundamental tendencies. The first consists of transforming the objective functions/rewards to embed safety measures. The second consists of modifying the exploration process through incorporation of external guidance and the use of a risk metric. It also summarized the emerging challenges down to those sub directions that are too approach-specific to enumerate here.

AI components are adopted onto autonomous systems as “perception and high intelligence” modules plugged on top of a base system platform that usually incorporates computerized control system and physical components. A deeper look at the electrical/electronic/mechanical architectures for such systems is needed [23, 77].

- **System Architecture:** Risk and uncertainty inherited within learning-enabled component is likely to be propagated and amplified into system-level. Hence, system architecture should incorporate new mechanisms of conflict resolution, fault-isolation, and fail-safe for each particular deployment of IA systems.
- **System Update:** Once the system is in the field, any modifications to the software would usually require the validation not just of the individual modifications but of the entire modified system. Revalidating the entire vehicle every time a relatively small level of learning occurs is not feasible with current methods of validation and system construction. Therefore the challenge is that incorporating any learning into a deployed system is not just a matter of regularly pushing software updates to the system code.

2.2 Levels of Autonomy

Autonomy in a vehicle —whether on the road or in the air—is not a binary property. Sheridan and Verplank [122] developed a guide for system levels of automation (Figure 1). The Society of Automotive Engineers [69] further derived this automation hierarchy into five discrete levels of automation specific to developing automated vehicles (Figure 2). These tables illustrate the varying levels of human management, supervision or oversight one may consider when designing for an intelligent system.

Increasingly autonomous (IA) systems lie along this spectrum of system capabilities from current automatic systems, such as autopilots and remotely piloted (non-autonomous) unmanned aircraft to highly sophisticated systems that would be needed to enable the extreme cases [34]. For this exercise we consider Automation Levels 8-9 and SAE levels 3–4 because in many ways they introduce more challenging problems. These semi-autonomous socio-technical systems operate in dynamic, safety-critical domains and require efficient and adaptable human-system interaction. Indeed, some autonomous driving researchers consider the human interactions of intermediate levels of autonomy intractable. Within this context, we specifically consider the automation providing the transition from two-pilot operations to a level of autonomy where an IA system reduces, changes or eliminates a human’s role.

Automation Level	Automation Description
1	The computer offers no assistance: human must take all decision and actions.
2	The computer offers a complete set of decision/action alternatives, or
3	narrows the selection down to a few, or
4	suggests one alternative, and
5	executes that suggestion if the human approves, or
6	allows the human a restricted time to veto before automatic execution, or
7	executes automatically, then necessarily informs humans, and
8	informs the human only if asked, or
9	informs the human only if it, the computer, decides to.
10	The computer decides everything and acts autonomously, ignoring the human.

Figure 1: Sheridan levels of automation

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/Deceleration	Monitoring of Driving Environment	Fallback Performance of Dynamic Driving Task	System Capability (Driving Modes)
Human driver monitors the driving environment						
0	No Automation	the full-time performance by the <i>human driver</i> of all aspects of the <i>dynamic driving task</i> , even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the <i>driving mode</i> -specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the <i>driving mode</i> -specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the <i>human driver</i> perform all remaining aspects of the <i>dynamic driving task</i>	System	Human driver	Human driver	Some driving modes
Automated driving system (“system”) monitors the driving environment						
3	Conditional Automation	the <i>driving mode</i> -specific performance by an <i>automated driving system</i> of all aspects of the dynamic driving task with the expectation that the <i>human driver</i> will respond appropriately to a <i>request to intervene</i>	System	System	Human driver	Some driving modes
4	High Automation	the <i>driving mode</i> -specific performance by an automated driving system of all aspects of the <i>dynamic driving task</i> , even if a <i>human driver</i> does not respond appropriately to a <i>request to intervene</i>	System	System	System	Some driving modes
5	Full Automation	the full-time performance by an <i>automated driving system</i> of all aspects of the <i>dynamic driving task</i> under all roadway and environmental conditions that can be managed by a <i>human driver</i>	System	System	System	All driving modes

Figure 2: SAE levels of automation

2.3 Challenges in Assuring Safety of IA Systems in Reduced Crew Operations

We focus on a scenario of reduced crew operation when an IA system is employed which reduces, changes or eliminates a human's role in transition from two-pilot operations. Currently, there are two pilots in a flight — a pilot flying (PF) and a pilot monitoring (PM). The PF is in charge of flying while the PM monitors the plane's instruments and displays, communicates with Air Traffic Control (ATC), and assists the PF. These roles are interchangeable, so both pilots must be competent enough to handle all tasks. Further, some tasks may be performed by both pilots or shared between them. Some tasks are done throughout the course of the flight, whereas others are done at frequent intervals or triggered by a specific events. Also, several tasks are inter-dependent and cascaded such as sensing a hazard and taking mitigation steps. Though replacing one of the pilots with an IA system has several benefits such as reduce costs and perform complex missions, such systems pose serious challenges in assuring its safety in the current well-established, and efficiently functioning aviation domain that is governed by rigorous rules.

2.3.1 The Challenge of Integration and Autonomy

Traditional aircraft automation consists of control systems (e.g., primary flight control, FADEC, autopilot, autothrottle) and fairly simple planning and execution engines (e.g., flight management, fuel management). These perform specific well-defined tasks, and are mostly localized to a single subsystem. Human pilots integrate all of these to accomplish the overall flight. Future IA systems will perform or assist or monitor many of these integrative tasks and this requires they have far greater 'knowledge' of the various aircraft systems, their interrelationships, and the overall system and aircraft architecture than any current system. Furthermore, they need knowledge of all the various tasks and procedures that are performed in accomplishing a flight, and need to maintain 'situation awareness' to identify those appropriate at the current time. This involves much more than the standard control functions (i.e., actually flying the aircraft) and includes activities such as operating radios, monitoring weather and so on.

Human pilots acquire the knowledge and skill to perform these tasks through training and experience, but much of it is tacit knowledge, organized and deployed using general intelligence. The challenge for IA systems is how to program comparable capabilities, and how to provide assurance for it. Broadly speaking, there seem to be three ways to accomplish the programming.

Explicit requirements: Develop detailed requirements for the various phases and procedures of flight and program these in a similar way to conventional aircraft systems.

Machine learning: Use machine learning to train an IA system based on the behavior of human pilots [17].

Model-based synthesis: Develop models of 'a complete flight' and derive IA behavior at runtime by automated planning and synthesis against this model.

It seems likely that explicit requirements and, especially, machine learning, could be ‘brittle’ in the face of previously unforeseen or un-encountered circumstances. The fact that autonomous agents can learn emergency procedures [16] does not change this assessment: they learn only the ones they have seen.

In all cases, there is the challenge of assurance. It seems to us that monitoring is the only feasible approach: an independent system should monitor the performance of both the IA system and the human pilot against a model of ‘safe flight’. This model would be simpler and more robust and easier to validate than models capable of synthesizing desired behavior: for example, a model for synthesis might contain complex logic for when the landing gear should be retracted, but a ‘safe flight’ model might simply specify that it should be retracted when more than 5,000 feet above ground level, or 250 knots airspeed, except in emergencies. The ‘safe flight’ model could be structured as a generic model plus a separate model for aircraft-specific parameters and customizations. The generic model could be developed as a common resource, managed and certified by international bodies or safety authorities, rather like (and derived from) the ‘Aviation Handbooks & Manuals’ maintained by the FAA [46].

A human pilot should not only be able to override IA actions or ‘safe flight’ monitor alarms, but to explicitly register ‘disagreement’. This would be used for subsequent analysis and improvement to the IA system and the ‘safe flight’ model.

An alarm from the ‘safe flight’ monitor should result in prompt action by the IA system or human pilot to rectify the situation or, for human pilots only, to override the alarm. If the alarm state persists, we enter the CRM or ‘never give up’ scenarios that are discussed below.

2.3.2 The Challenge of CRM

CRM is ‘Crew Resource Management’ and refers to the teamwork approach to flight safety and problem solving that has been recommended since the United flight 173 crash in 1978. It is a human factors issue concerned with situation awareness, communication, and allocation of responsibilities rather than technical topics. In an IA aircraft with reduced crew, the IA system must engage with the human pilot in CRM just as an all-human crew would.

One aspect of this is that in some circumstances the IA system might need to offer tasks beyond those certified (or certifiable): e.g., with no air data (as with Air France flight 447 in 2009 [45]), it might offer to hold the aircraft straight and level while the pilot troubleshoots; this topic is discussed below under Never Give Up.

More germane here is that the IA system might need to explain its reasoning to a human pilot, or to persuade a human pilot that they are wrong. There is much current research that aims to develop ‘explainable AI’ but this typically assumes that the human participant is a neutral observer, a ‘blank slate’, who needs merely to be informed how the AI system reached its recommendation. In many

aircraft crashes, the pilot flying forms an incorrect assessment of the situation and sticks to it despite overwhelming indications to the contrary, and despite attempts by human crew to point this out (e.g., Air India flight 855 in 1978 [114]), so the task of an IA system is much greater than merely offering explanations.

Assurance for CRM seems particularly challenging as it concerns human interaction. There are certain principles that can be checked by monitoring (e.g., at any time, either the IA system or the human pilot must have accepted responsibility for flying the aircraft), and the pervasive monitoring for ‘safe flight’ also provides some guarantees. But productive dialog between human pilots and IA automation seems to be something that can be validate only through simulation, experiment, and experience.

Human-IA Interaction

Removing a member of the flight crew necessarily involves deep human interaction. Many of the tasks we describe are commands by the pilot flying to tell the pilot monitoring to adjust automation settings, engage or disengage modes, deploy landing gear, etc. This suggests a ready-made set of super instructions where the pilot issues a command that triggers a series of actions and subsequent monitoring tasks. On the other hand, the pilot issues these commands to a person they trust is trained and capable to carry them out. This trust is a significant challenge for an IA system. Our task list poses these interesting human-automation interactions:

- Querying the pilot for a questionable command or action.
- Providing a rationale for any action not directly commanded by the pilot.
- Indicating current mode or role for the duration of the mode.
- Coordinating when the pilot and IA system have different versions of procedures, manuals, etc.

An essential reason—possibly the reason—why two pilots are required in the cockpit is to provide second opinions. Most of the time, there is a standard procedure and the pilot acts with immediate agreement from the PM. Once any abnormal situation arises, however, the pilots may need to confer about the appropriate action. An IA system needs to be able to question a command or action, and it needs to supply a rationale for why it may be incorrect. In an emergency, there may not be time to process a significantly different scenario than the pilot was already considering. This means the system needs to keep the pilot informed throughout the flight of how it perceives everything is progressing. Asserting its own understanding of the situation may not be sufficient, however; it may also need to understand the pilot’s concept of where they are and how they got there.

One of the requirements categories involves checking reference material. This presumably is loaded into the IA system and updated regularly. These materials may revise SOP for efficiency and cost savings for the airline, or they may include FAA-mandated safety changes. It is possible that a pilot has not been briefed on changes that have been loaded into a particular IA system, or that a pilot is more current than the system. In the latter scenario, we would expect the pilot to understand the actions of the system. It may question a command or take an action that used to be standard. A pilot should be able to understand where the disagreement comes from and override appropriately.

Expecting the IA system to recognize that a pilot is operating from an older manual or SOP and what it should do is an important question. Should the system defer to the pilot and revert to the older version? For safety-related changes, the IA system may need to inform the pilot of the new instructions. The pilot then needs to trust the system is correct and has the most current data.

Trust in Automation

In many cases with IA, human pilots will become relegated to a more supervisory/administrative role, a passive role not conducive to maintaining attentional engagement. This situation breeds overreliance on the automation, and causes pilots to become less engaged in the human-IA interaction, potentially losing awareness of the mode under which the system is operating. Increasing complexity of the system and automation modes may lead to poor understanding of the interaction between the IA and the state of the aircraft or phase of flight. This can induce pilot confusion and an erosion of trust, which often leads to inefficient and possibly hazardous decision making.

The concept of trust plays a critical role in the design and use of automated systems. If operators trust the automation they will use it more consistently. If their trust drops below a particular threshold the operator will override it and may discontinue use. If operators override the automation too frequently, the system will be less efficient. Conversely, too much trust can lead to overreliance, complacency and failure to detect hazardous conditions. Thus, the operator's trust in an automated system has a direct impact on its performance and safety.

2.3.3 The Challenge of Never Give up

Traditional aircraft automation disconnects when it detects situations beyond its certified capability (e.g., too many faulty or inconsistent sensor readings) and the human pilots are expected to save the day. An IA aircraft with reduced crew cannot do this; the single pilot may need to direct their focus elsewhere (e.g., fighting a cockpit fire, troubleshooting). Thus an IA system must have a 'never give up' (NGU) capability and this may mean that it must fly the aircraft in conditions that are beyond what is certified, or indeed certifiable. An example is Air France flight 447 of 2009 where ice crystals blocked the pitot tubes, depriving the aircraft of air data. In high altitude cruise there is little margin between stall and mach buffet and no safe way to fly the aircraft in the absence of airspeed data. The recommended procedure for human pilots is to maintain constant pitch and thrust. An IA aircraft could offer to do this (and might be able to do a bit better using stale air data and live groundspeed) but it cannot be certified that it is safe to do so. Another example would be United flight 232 of 1989 (Sioux City) where all hydraulics were lost and the pilots controlled the aircraft with differential engine thrust. An IA aircraft could use an adaptive controller to do this, but it could not be certified as safe.

It seems to us that certification requirements should be adjusted for NGU situations. It also seems to us that the larger air traffic and airspace management system needs to explicitly acknowledge these

situations. For example, an IA aircraft operating in NGU mode should be treated as if a human pilot had declared an emergency. Similarly the goals of the IA system should change so that instead of continuing the nominal mission it attempts to land as soon as possible.

Reduced certification requirements for NGU situation seems uncontroversial, but there are two complications. One is to ensure that this does not provide a ‘get out’ for difficult cases that do nonetheless demand strong certification, and the other is to provide assurance that NGU modes are not entered inappropriately. The latter was a significant concern for previous research programs, but it seems to us that the pervasive monitoring against ‘safe flight’ models that is required for IA systems could provide the necessary assurance.

2.3.4 The Challenge of Unconventional Implementations

As we noted in our requirements report, current guidelines for assurance of airborne software require predictability and determinism, and this is at odds with the kinds of software that is envisaged for IA systems, which will employ techniques based on search such as model-based diagnosis and planning, and also those based on machine learning. Search-based methods lack determinism and predictability since unfortunate choices can lead to protracted searches and timeouts, but assurance for the correctness of successful searches seems relatively straightforward: the searches are conducted on a model, so correctness rests on validity and completeness of the model, and that seems a tractable problem, as we discuss in the following section.

Machine learning poses more difficult challenges. Changes to a few pixels can cause image classification systems to abruptly change their classification (e.g., from ‘stop sign’ to ‘birdcage’); [71] gives examples. These ‘adversarial perturbations’ can be deliberately constructed and to humans the resulting images appear the same as the originals—suggesting that the machine classifier is responding to completely different features. Analysis of systems generated by learning is complicated because the implementation often takes the form of deep neural nets, and these are opaque to most forms of inspection and review. However, there is much current research aimed at causing machine learning to generate more robust and understandable systems, and there has also been some success in formal analysis and verification of these topics (e.g., [71, 80]), so these are areas where we can expect rapid change.

New techniques might allow assurance for the correctness of systems and classifiers generated by machine learning, but these would be predicated on the assumption that the live situation is covered by examples in the training set. A promising idea is to learn a compact representation of the training set using a simple and sound type of learner [134]. This would allow runtime determination whether the current situation is ‘inside’ or ‘outside’ the space of examples encountered in training.

We recommended that *Pervasive Monitoring* seemed the most appropriate assurance response to those challenges and we explored that technique in subsequent reports. However, we do not advocate that pervasive monitoring is the only assurance method deployed: assurance for individual functions

should also be provided by suitably extended versions of traditional methods for verification and validation. We say ‘suitably extended’ because it is expected that IA aircraft will employ unconventional implementations (the fourth challenge above) such as machine learning. The difficulties due to machine learning are that a) the implementation (e.g., a neural net) is opaque, so there is no design or code to analyze using traditional methods, and b) there is typically no conventional requirement against which traditional validation or verification can be performed, merely a ‘training set’ of examples.

2.4 Shortfalls of Traditional Assurance Approach for Future IA Systems

The traditional assurance approach prescribes fixed V&V process objectives with associated activities via guidance provided in ARP4754A [127], DO-254 [40] and DO-178C [104]. These approaches have worked well for the assurance of traditional avionics and control systems, cockpit displays, and flight management systems and software. However, as indicated by recent research [39], the traditional methods may already be under stress, given the complexities of modern distributed systems, and the associated multi-tier supply chains. Given, projected complexity increase associated with the next generation of IA system, it is likely that additional assurance methods and technologies will be required.

The following are the salient aspects of this approach and the potential shortfalls with approach for IA systems:

1. There is an ‘implicit’ assurance case built into DO-178C process activities and associated verification approaches and metrics that start from high-level requirements and proceed to design and software implementation [68]. This has worked for traditional types of control, logic-based algorithms and displays, based upon decades of experience with those types of system architectures and the types of functions implemented in software.

Potential Shortfalls: However, there will be new types of technology (e.g., rule-based/Bayesian reasoning, machine-learning) used to implement IA-System functionality that will not be amenable to the DO-178C/DO-254 process activities and associated verification objectives. In particular, metrics for data-control coupling, requirements based test coverage, and modified condition/decision coverage for software structure will not be directly applicable. Even if the underlying hardware is assured in accordance with DO-254, and neural network software libraries are assured in accordance with DO-178C, the assurance is insufficient to address the emergent properties derived from the network weights. Therefore full structural coverage of a neural network implementation, may yield little insight with respect to the correctness or completeness of the learned knowledge and generalizations, encoded within the network topology and weights. While the broad principles of requirements-based verification still stand, we believe that more flexible approach based on assurance cases will be needed. New techniques such as [71, 80] may yield more meaningful verification data.

2. In current human-piloted aircraft, for many automation systems such as autopilots, there is an implicit assumption that disconnecting the automation and yielding control back to the human pilot is a safe action. Hence, the safety assurance is largely predicated on humans taking control when the automation (e.g., autopilot) makes an error, doesn't perform according to human pilot's expectations, or just gives up. The functional hazard analysis (FHA) and the system safety assessment (SSA) for the most part don't address these issues. Recent incidents, such as the demise of Air France flight 447 [27] may indicate that such assumptions may be more complex when the complexities of modern cockpits are reassessed.

Potential Shortfalls: With the use of IA systems, more responsibility for all aspects of 'flying' will be assigned to IA system, enabled by more 'intelligent' capabilities coming in IA. This may change the dominant 'fail-stop' mind set towards 'fail-operational' model. When combined with the increased exposure time, requiring the automation to function correctly for the entire flight duration, we expect that such requirements may have significant impact on system architecture.¹

In addition the new interaction patterns, communication and hand-off protocols of the human and the IA systems will need to be addressed. For example in reduced crew operations, IA system may take on the roles of flying or monitoring pilot, interchangeably with humans [8]. This introduces more complex patterns of safety arbitration and hazards that cannot be handled by traditional methods, which are largely focused on component failures within the automation system.

The situation is further exacerbated by the pace of technology and architecture evolution that will most likely be associated with IA systems. The traditional assurance processes have largely been assisted and informed by the decades of experience gained from fielded systems. In turn, standardized architectures, such as the dual-lane FADEC, have been evolved to satisfy the needs of these target systems. In certain domains, such as engine control, the reliability of electronic and software based systems have led to revised certification guidelines [14] that allow these systems to operate, for a limited duration with in field faults. These arguments essentially based on the fleet experience and validated predicted reliability of the engine systems. It should be noted that such arguments are largely assisted by the limited scope of the systems, for example the influence of a single engine. When one contrasts this authority with the broader influence of potential cockpit wide IA automation, the traditional historically informed approaches may not be applicable. In addition, as discussed above the new complexities, new interaction modes, and new types of capabilities brought by IA systems will also introduce new challenges.

3. In the traditional approach to assurance, the regulatory, maintenance, and other processes auxiliary to the operational system are implicitly considered in the activities to assure the safety of the aircraft operation. These processes occur continuously during the operational life of an

¹Often the assumed exposure time assumed for critical system failure may be limited to a short period, such as a few minutes of the assumed landing interval duration, that is associated with the hazard of interest

aircraft fleet and include: Advisory Circulars, Airworthiness Directives; Standard Operating Procedures (SOP), changes in flight rules and flight manuals; bug fixes and upgrades of aircraft systems/software functions, and their regular maintenance.

Potential Shortfalls: The traditional approach to consider these processes implicitly in safety assurance has several shortcomings for applicability to future use of IA systems. As we have stated above, IA systems will become increasingly more ‘intelligent’ and share flying and safety responsibilities with human pilots. While the dissemination and implementation of operational changes due to the auxiliary processes is currently handled by humans, in future, both humans and IA systems will share this responsibility. To assure that the IA systems properly execute this responsibility, including upgrades to the IA systems to handle changes required by these processes, all relevant aspects of these processes will need to be modeled explicitly in terms of flow of information/updates through process steps and the impact on safety. Furthermore, there are different potential paths/steps in this information flow from the source of information (e.g., flight rule change, maintenance advisory) to human pilots and IA system. This can lead to an inconsistency between the world-views of humans and IA system and adversely impact safety.

4. In current human-machine interaction scenarios in the cockpit, the interactions are limited to commands, mode setting for auto-pilots, interaction with flight displays and instruments, and recent use of touch screens for certain interactions. Due to the limited scope of such interactions, the regulatory guidance for safety assurance aspects is quite limited.

Potential Shortfalls: In future scenarios of employing IA systems in reduced-crew operations, human-machine interaction scenarios will have complex patterns and quite enhanced levels of the semantics of information exchange due to more ‘intelligent’ IA capabilities participating in crew resource management (CRM). Without additional considerations being applied towards assurance, such complex interactions can be much more susceptible to mode confusion and automation surprise than current systems. In particular, there is a need to clearly delineate human vs. automation roles in each task and define functional and performance requirements with appropriate specificity for each activity in human-IA interactions.

3 Safety Assurance Approach for IA Systems

To address the challenges and potential shortfalls of traditional methods described in Section 2, several aspects of safety assurance must be addressed. In this section we present the overall safety assurance context for an IA system and various aspects that must be considered, leading to a framework for an assurance case.

Subsection 3.1 describes the overall assurance context and aspects of assurance that apply to IA systems. This also provides a roadmap to the rest of this report—the later sections in this report each address specific aspects of assurance within this context. Subsection 3.2 provides rationale for using an assurance case that will rely on property-based assurance in large part. Subsection 3.3 presents a framework for an assurance case, illustrating the types of arguments.

3.1 Safety Assurance Context and Aspects

Figure 3 depicts the broad context of assurance that needs to be considered for IA systems. The development and operational processes along with the operational system provide the context for safety analysis in terms of sources of hazards and system functional requirements. The safety analysis provides a systematic examination of hazards, causal factors, and mitigation approaches. We posit that a pervasive runtime monitor integrated with the architecture can be an effective means of mitigation. The V&V objectives and activities provide additional design-time means of mitigation for system development artifacts. Finally, an assurance case assembles the arguments and evidence across all the aspects within this overall context.

We briefly describe below the safety assurance aspects depicted in Figure 3:

Hazard Sources. The system itself and all the process elements that impact the system behavior constitute sources of hazards posed by system operation—any failure in a source can lead to a hazard. The *product* includes the aircraft and parts of its operational context (e.g., crew interactions including automation, communication with ATC, certain other interactions). The regulatory guidance includes certification guidelines/standards, advisory circulars, airworthiness directives, flight rules, standard operating procedures (SOP), etc. The *development process* consists of all activities that start from system requirements and result in the implementation of the system. The auxiliary operational processes consist of activities involving operational configurations, maintenance, and dissemination of changes in regulatory guidance including flight-rules, SOP, and external operational data such as maps, flight plans.

Functional Requirements. The system high-level requirements define the functions the system performs and allocation to IA components. A critical consideration here is the sharing of responsibilities between a human pilot and an IA component within a Crew Resource Management (CRM) context, which introduces new failure modes in terms of human-machine interactions. These requirements are elaborated in Section 4.

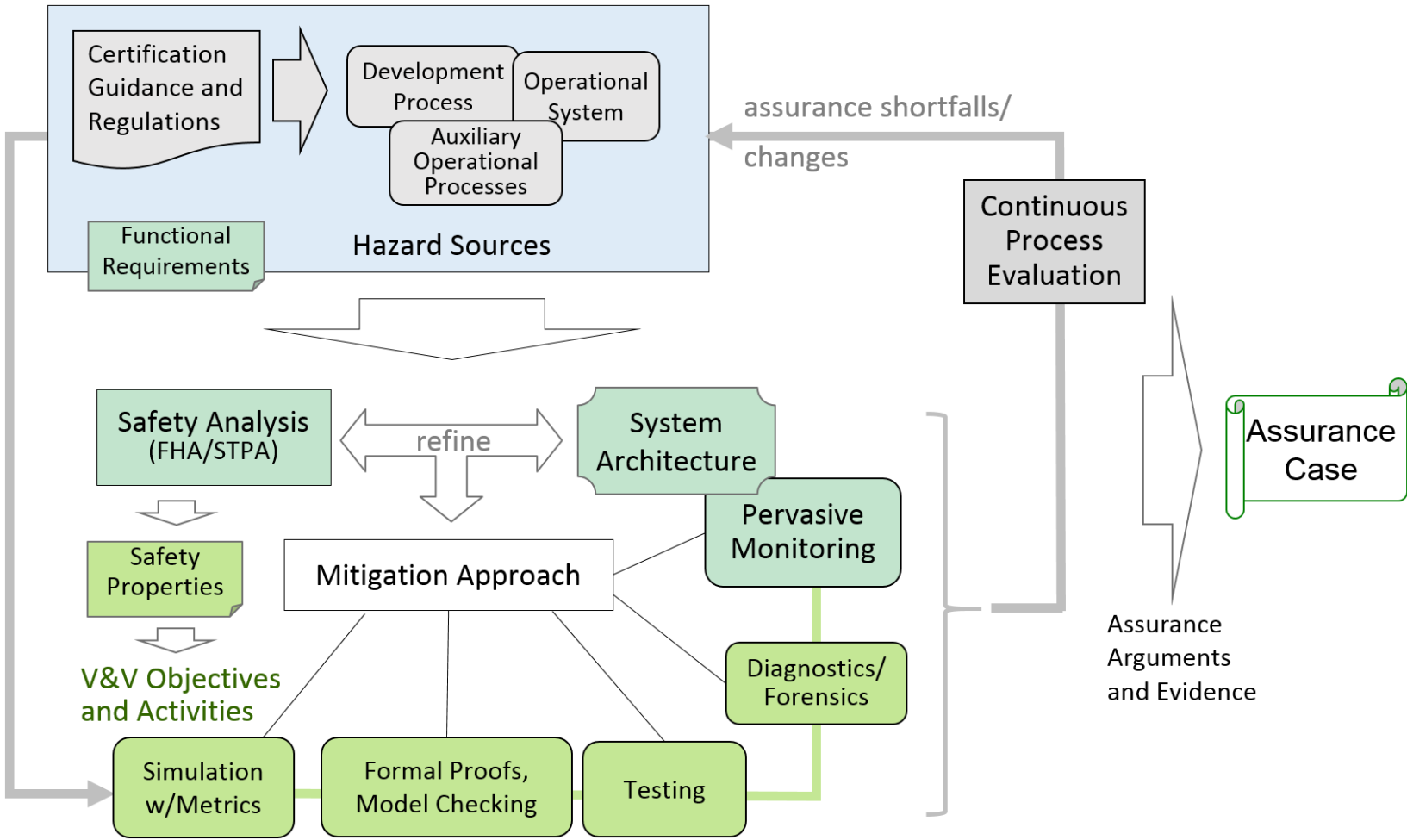


Figure 3: Context for Safety Assurance

Safety Analysis. A comprehensive safety analysis must address all sources of hazards in the processes described above. For the hazards sources in the product (system), a new failure modes in terms of human-machine interactions and the necessity for IA to “never give up” and take final responsibility for safety. Other new types of hazard sources are introduced in the auxiliary operational processes which are implicit (not addressed) in the current system. Given such a scope of hazard analysis, we believe that Systems Theoretic Process Analysis (STPA) method, with suitable extensions, would be beneficial here. We elaborate further on this in Section 5.

System Architecture and Pervasive Monitoring. System architecture is central to safety assurance in conjunction with safety analysis. The architecture provides fault-tolerance to mitigate a variety of hazards arising from component failures—with safety analysis guiding iterative architecture refinement—as elaborated in Section 6 and further in subsection 6.2. We also suggest that pervasive runtime monitoring may be the most effective and credible method for assurance of certain critical aspects such systems, combined with static V&V methods as appropriate. We provide additional rationale for this in subsection 2.3.4 and further elaborate in subsection 6.3.

V&V Objectives and Activities. V&V activities are performed as part of the development processes to provide assurance about the design and implementation of the product. Simulations that are directed towards assessing advanced automation capabilities and exercising scenarios of human-machine interaction will play a central role in the V&V, as described in more detail in Section 7.5. Other techniques include use of formal proofs, model checking, static analysis, and testing that have been the hallmark of V&V in current systems based upon DO-178C guidance [104]. For advanced automation functions based upon AI techniques such as machine learning, new design-time V&V approaches will need to be employed. We elaborate further on this in Section 7.

Continuous Process Evaluation. We suggest that a continuous evaluation of the entire assurance process is needed due to rapidly evolving AI technology (e.g. machine learning), increasing automation role, and changing assumptions about the regulatory guidance, auxiliary operational processes. The evaluation results are then used to update all parts of the process, hazard analysis, and V&V capabilities to address the assurance shortfalls though the operational life of a fleet.

Assurance Case. We believe that an assurance case is the most suitable method to formalize the arguments and evidence from this broad/diverse context of hazard sources and mitigation approaches—providing a basis for assurance and certification of IA systems. We elaborate further on the rationale in the next subsection 3.2 and discuss a framework in subsection 3.3.

3.2 Rationale for Assurance Case and Property-Based Assurance

Assurance for aircraft systems and software is currently based on guidelines such as ARP 4754A [127] and DO-178C [104]. As we have discussed earlier in Section 2.4, future IA systems will not conform to some of the assumptions on which those guidelines are based. Hence, we recommend their certi-

fication should build on the idea of assurance cases. An assurance case uses explicit argument and evidence to provide justified belief in a claim concerning safety (or other property) of a system. Any credible approach to assurance must surely employ similar concepts; the novelty in assurance cases is that they are made explicit. In contrast, guidelines such as DO-178C, for example, specify only the activities to be performed and the evidence of this accomplishment—the argument is left implicit [68] (but presumably informed discussion and debate in the committee that produced the document).

Assurance case techniques have been developed and applied in various contexts for a number of years now. Rinehart et al. [105] provide a good survey of assurance case applications across the industry. They analyze, through interviews with stakeholders, the business/operational contexts regarding the applicability and benefits of assurance case over traditional methods. To manage the mapping of assurance methods and organizational responsibilities, two hypothetical subdivisions of assurance case are suggested: “Development Assurance Case” and “Integrated Operations Assurance Case.”

For IA systems, however, we believe that the context and subdivisions of the assurance case will be much broader—as shown in Figure 3. The composite assurance case will need to assemble various types of arguments and evidence—using diverse sub-contexts, methods, and tools. We expect that the recent FAA effort on “overarching properties” [39] will allow the freedom (from prescriptive methods) for formulating an assurance case that can be structured along the lines outlined here to fully address the challenges posed by the use of IA systems.

3.2.1 Property-Based Assurance.

Current practice based on guidelines including ARP 4754A and DO-178C/DO-178C has produced safe aircraft, but it is not really understood what aspects of these practices deliver this outcome. Hence, new methods based on assurance cases cannot yet guarantee effectiveness. A recent paper [112] provides an idealized description and interpretation for assurance cases that shows how they can provide the support necessary to justify certification against the requirement that a catastrophic failure condition “. . . is not expected to occur during the entire operational life of all airplanes of one type” [49].

The idea is that the assurance case provides confidence that the software is free of faults and this confidence can be expressed as a subjective probability, which then translates into a probability that the software will suffer no failures in its entire operational lifetime. However, there is a possibility that our confidence is misplaced and so we also need to be sure that the system will not fail even if it does contain faults. Confidence in that property is acquired incrementally through flight tests and operational experience using a Bayesian analysis (from [130]) that is bootstrapped from the initial confidence in fault-freeness under worst-case assumptions, and is therefore conservative.

The assurance case argument is partitioned into *evidential* steps, which combine items of evidence to deliver probabilistic confidence in local claims, and *reasoning* steps that conjoin these to justify higher level claims and, ultimately, the top claim. Evidence supporting a local claim may be interdependent, but separate local claims are assumed to be independent. An important point is that to

provide adequate confidence in its top claim, the argument steps of an assurance case must be *indefeasible*, meaning we must be so sure that all contingencies (i.e., hazards to the system and defeaters to the argument) have been identified and considered that there is no new information that would change our assessment.

IA aircraft systems pose challenges to traditional assurance practices; these systems can range over a wide spectrum from those that do new things in a fairly traditional way (e.g., increased automation but developed from conventional requirements through traditional engineering methods) to those that employ methods of artificial intelligence (e.g., machine learning, model-driven adaptation).

We believe that assurance cases have the flexibility to ensure safety for all of these and so we explore the full spectrum in the following sections. However, we suspect that the most plausible IA systems will be from the more conservative end of the spectrum and we will develop this in more depth in later stages of the project.

The purpose of assurance is to provide guarantees about the future, so that we can be confident our system will do what we want and will not cause or allow bad things to happen. These guarantees are based on two bedrock assumptions: first, that it *is possible* to predict the future behavior of our system in the world and, second, that with enough effort we can obtain *near-omniscient* (i.e., indefeasible) insight into those behaviors.

Those bedrock assumptions are credible only in highly constrained circumstances: those where both our system and the world in which it operates are fully predictable. These constraints do apply to traditional aircraft systems where the world consists of well-understood physical phenomena (nothing is more accurately modeled than the flow of air over a wing), simple physical devices (flaps, landing gear, etc.), other computerized systems, and trained and disciplined human operators (pilots and air traffic controllers).

But even in this simple space, additional constraints are imposed to ensure predictability. For example, the FAA requires aircraft systems to be deterministic. This affects software design (e.g., choice of scheduling strategy), hardware exploitation (e.g., use of caches and multiple cores), and algorithm design. For example, control laws for the ailerons and other control surfaces need to change as the aircraft transitions from slow flight in warm dense air at takeoff to transonic flight in cold thin air at cruise. A single adaptive control law could do this, but these are considered insufficiently predictable so instead as many as 30 separate, individually certified, control laws are used with complex rules to smooth the transitions between them. Concern about predictability is not unreasonable: adaptive control is implicated in the crash of one of NASA's X-15 space-planes and the death of its pilot [44] (the problem was not in adaptive control considered alone, but its behavior in the presence of a failure, as one of the elevons had lost 80% of its control effectiveness).

The constraints that ensure the bedrock assumptions are becoming less tenable in modern systems, so on what alternative foundations can credible assurance be constructed? There are two dimensions to this question: one concerns properties and predictability (a strong property may not be indefeasibly

predictable, but a weaker one might), and the other concerns the location of unpredictability: is it in the system (where unpredictability can sometimes be managed by architectural means), or in the world, or both?

An example where unpredictability in the system is transformed into predictability for a weaker property is provided by the so-called “simplex” architecture [121]. Here, we have a sophisticated primary component that provides attractive but unassurable behavior (e.g., an adaptive control law), and a secondary component that is less attractive, but assurable (e.g., a verifiably safe, but crude control law). The secondary component guarantees safety as long as the system is within some envelope; the primary component generally drives the system but the architecture transitions to the secondary component when monitoring indicates the system is approaching the edge of the secondary component’s envelope.

The simplex architecture can be seen as the extension to reactive systems of an earlier technique for sequential systems called (provably) “safe programming” [12], and both are related to the “detectors and correctors” treatment of fault tolerance [13] and similar treatments for security [134].

All these techniques depend on a monitor function that detects incipient failure of the primary. A monitor can be very simple: it need not to do anything active, just observe the values of sensors and internal variables and periodically evaluate some predicate. The predicate need not be a full check on the system’s operation, but rather some property that is relevant to its safety claim. Due to its simplicity, it is plausible that assurance can deliver strong confidence that a monitor is fault-free with respect to the monitored property. Now it is a remarkable fact that when we have a system with two “channels”, one that is reliable with $pdf P_A$, and one for which we have confidence P_B in its nonfaultiness, then the reliability of their combination has $pdf \leq P_B \times P_B$ [86].² Notice that the $pdfs$ of two reliable channels *cannot* be multiplied together because their failures may not be independent.

The simplest arrangement of this type is a “monitored architecture” where the first channel provides the standard operation of the system and the second is a monitor that signals failure when its property is violated. This prevents transitions into hazardous states and notifies higher-level fault management (e.g., human pilots) that something is amiss. In a monitored architecture, we must consider failures of commission (i.e., the monitor signals failure when it should not) as well as omission (i.e., the monitor does not signal when it should). These details are developed and analyzed in the paper cited above [86] and illustrated for the case of an aircraft incident that involved failure of a massively redundant fuel management system. The notable conclusion is that confidence in nonfaultiness of the monitor provides strong assurance for reliability of the overall system. The closer the monitored property is to a top-level claim, the stronger and more direct this assurance will be.

The simplex architecture and its variants extend monitored architectures by adding a secondary operational channel that can take over when the monitor detects failure of the primary. Assurance for

²This is an *aleatoric* analysis; to apply it, we need *epistemic* estimates of the parameters involved and these may introduce dependencies. However, it is often possible to manage these by making conservative estimates [143].

this architecture can be based on that for the monitor and the secondary channel alone, with little or no reliance on the primary operational channel. (Presumably the primary channel delivers attractive behavior, so weak assurance for the ordinary requirements may derive from this channel, while strong assurance for safety derives from the monitor and secondary channel.)

The simplex architecture can be extended to multiple levels of fall-back with less and less desirable, but still safe, backup channels or behaviors. But what if there is no safe behavior? The crash of Air France flight 447 in 2009 provides an example: here, the Pitot tubes were blocked by ice crystals, depriving the airplane of airspeed data. The autopilot detected the problem and disconnected, handing control to the pilots. In the absence of airspeed data, pilots are instructed to hold pitch and thrust constant, although this is difficult (AF 447 was in turbulence and cloud) and delicate (at altitude there is a small margin between stall and Mach buffet/overspeed). However, the pilot flying misinterpreted the situation and stalled the airplane all the way down to the ocean, resulting in the deaths of all on board [27]. The autopilot of a future, “more autonomous” aircraft, could offer to hold the plane straight and level while the pilot troubleshoots. (The autothrottle of AF 447 did automatically enter thrust lock, but the pilot flying disabled it.) But what would be the basis for assuring the safety of this procedure? In the absence of reliable airspeed data, it is impossible to give guarantees on the safety of flight.

Assurance requires infeasible confidence in strong claims for critical properties; here, we cannot achieve this, so should we relax infeasibility or the strength of the claims? Our opinion is that we should retain infeasibility—we must remain sure that nothing has been overlooked—but lower the threshold on evidential claims. In the absence of air data, we cannot be sure of the performance of the wing, but we can attempt to estimate that data from other sensors (e.g., ground speed and altitude from GPS) and recent history (e.g., stale air data), with fixed (or table-lookup) pitch and thrust as a final fallback. Although these estimates cannot support strong assurance, it seems to us that this approach is the best we can do and it retains the basic outline and justification we have developed for trust in assurance cases.

Monitoring provides a systematic way to deal with unpredictability: if we cannot predict how our system or the world will behave, we monitor them and base our assurance case on claims that are guaranteed *assuming* the monitored properties. However, this requires that we can accurately monitor properties of interest, which may not be so, especially when uncertainty is located in the world. Automated driving provides an instructive example; a comparable aircraft example would be automated taxiing and other ground operations. Here, we have cameras and possibly lidars and other sensors, and a sensor fusion and vision understanding system that attempts to discern the layout of the road and the presence of obstacles and other hazards. The vision system uses machine learning, so its operation is opaque and unsuited to conventional assurance and we have no independent way to monitor its correct interpretation of the scene. However, although we cannot monitor a safety claim directly related to its top-level function, we can monitor related properties. For example, given the car’s location (from GPS) and good map data, we can calculate static features of the expected scene,

such as a gas station on the left and a mailbox on the right. A monitor can calculate these predicted features and check that the vision system reports them. Such a monitor does not guarantee that the vision system is operating safely, but at least we know it is awake and interpreting the world around it with some accuracy [91, 129].

Beyond monitoring and backup channels to ensure satisfaction of assumptions, safety properties, or other weaker claims, are systems that adapt their behavior to ensure these. This can be appropriate in highly unpredictable environments, such as those involving collaboration with humans, as in IA systems where the automation may replace one of the pilots. In addition to mandated and airline-specific standard operating procedures (SOPs), there is a huge amount of “implicit knowledge” underlying the interaction of human pilots. Attempting to program suitable behavior for each scenario into an IA system seems difficult and fault prone. It might be better to monitor the total system state and pilot activity against a model of the aircraft, SOPs, checklists, and high-level “rules of flight” (i.e., safety requirements) and to employ alternative courses of action when the state does not evolve as expected. For example, if the IA system operating as the pilot flying (PF) requests the human as pilot monitoring (PM) to deploy the landing gear, it should expect to see the state equivalent to “three greens” (gear down and locked) within some reasonable time and, if not, take some alternative action (e.g., repeat the request, try to diagnose the problem, release the gear itself, initiate a go-around, switch roles etc.).³

Some expect these behaviors to be pre-programmed and propose to verify their scripts against the “rules of flight” [51, 137], whereas others advocate their generation at runtime by an automated planner or other synthesis procedure driven from those rules and the modeled environment (among other benefits, a system of this kind can explain the reasons for its actions). Assurance then derives from a combination of the validity of the rules (assured statically, at design time), and their interpretation and monitoring at runtime. Because some of the assurance is derived from runtime analysis, this approach is provocatively named “runtime certification” [109]. Provocatively also, we can speculate that advanced and future systems of this kind might evaluate their actions not merely against SOPs but against models of social norms and ethics (already there is an annual workshop on “Social Trust in Autonomous Robots”).

It is unlikely that autonomous systems operating in unpredictable environments can be assured to the level of today’s highly constrained systems. A plausible goal would be “at least as good as a human”; in the case of automated driving this might be quantified as more than 165,000 miles between crashes.⁴

³Inadequate monitoring and challenging are implicated in the majority of aircraft accidents attributed to crew error [53].

⁴To be clear, we would expect classical car and aircraft systems to be assured to the level they are today, or better; it is their automated driving and “more autonomous” autopilots that might be assured to the human-equivalent level. Note also that a crash by an automated car should lead to investigation of the precipitating scenario and improvement in the automated driving software that benefits the safety of all automated cars, whereas human drivers learn only from their own experience.

As in cases where failures render the system unsafe (e.g., the AF 447 scenario described earlier), assurance cases for autonomous systems operating in unpredictable environments cannot be watertight. So we have to decide whether some alternative means of assurance should be adopted (e.g., licensing, or massive testing) or, if assurance cases are retained, whether we should relax the infeasibility criterion or other elements of the traditional case.

In our opinion, the analysis that relates assurance to confidence in absence of faults and thence to absence of failures retains its value even when assurance goals are relaxed (i.e., we reduce the duration for which we expect no failures), because some confidence in absence of faults is needed to “bootstrap” the Bayesian analysis that demonstrates the value of (massive) pre-deployment testing and subsequent experience in operation. An assurance case is surely the only intellectually credible means to obtain that confidence: what other basis can there be apart from evidence and argument? Similarly, infeasibility seems the only credible basis for justified belief: unless applicants and certifiers have made a comprehensive attempt to think of *everything* that can go wrong with both the system and the argument (i.e., all hazards and defeaters), then they have failed in their duty.

Now, in an unpredictable environment, there could be some “unknown unknowns”, and it may be acceptable to represent this in a “catchall” subclaim that acknowledges unknown defeaters and thereby “fakes” infeasibility. Of course, no evidence can be provided for these unknowns (or they would not be unknown) so the case will have an ugly and visible incompleteness, but this as it should be, and is preferable to simply bypassing the problem by abandoning infeasibility—although a better course is to improve the system or its assurance case so that no unknowns remain.

The previous paragraphs provide a tour of the challenges and some of the responses in providing assurance for IA systems. Our general conclusion is that assurance cases provide a flexible framework that is adequate to the task. The way in which IA systems differ from conventional aircraft automation is that their internal operation, and their external environment, are less predictable. Assurance can be achieved, despite this uncertainty, through use of runtime monitoring and suitable architectural accommodations.

The most difficult challenge may be to achieve and provide assurance for a suitable form of “liveness” or continued operation in the face of difficulties. At present, automated aircraft systems give up at the first sign of difficulty and return control to the pilots. Proposed IA systems will assist pilots in more demanding “cognitive” tasks, but their likely means of assurance is through static or runtime checking against SOPs and “rules of flight”, with reversion to fully manual operation if a violation is detected. Such a reversion could be far more difficult for a human pilot to handle than those of current systems since the IA system will have assumed some of the cognitive burden and the human pilot’s situation awareness regarding those aspects may be correspondingly low. Thus, a system that provides increased autonomy must also accept increased responsibility and cannot simply revert to human control when difficulties are encountered. There may then be a choice between two difficult technical challenges: providing assurance for safe continued operation of a conventionally programmed system in “almost all” circumstances, or shifting to an adaptive system whose behavior employs runtime

synthesis and whose assurance may likewise depend on runtime calculation.

3.3 Framework for IA Assurance Case

We discussed the context and rationale for using an assurance case as the basis for assurance and certification of IA systems in subsections 3.1 and 3.2. The basic idea is that an assurance case uses a *structured argument* to provide justified belief in a *claim* (typically about safety) concerning a system based on *evidence* about its design and construction. We argue that the justification should be *indefeasible*, which means there is no (or, more realistically, we cannot imagine any) new evidence that would cause us to retract our belief. Indefeasibility is our criterion for ensuring that the case is complete: that is, it has no “holes.”

A structured argument is a hierarchical collection of individual *argument steps*, each of which justifies a *local claim* on the basis of evidence and/or lower-level subclaims. A trivial example is shown on the left in Figure 4, where a top claim C is justified by an argument step AS₁ on the basis of evidence E₃ and subclaim SC₁, which itself is justified by argument step AS₂ on the basis of evidence E₁ and E₂. Assurance cases often are portrayed graphically, as in the figure, and two such graphical notations are in common use: Claims-Argument-Evidence, or CAE [1], and Goal Structuring Notation, or GSN [81]; however, Figure 4 is generic and does not represent any specific notation. The boxes in the Figure contain, or reference, descriptions of the artifacts concerned: for evidence this may be substantial, including results of tests, formal verifications, etc., and for argument steps it will be a narrative explanation or “warrant” why the cited subclaims and evidence are sufficient to justify the local parent claim.

To make this concrete, in a conventional software assurance case, the claim C will concern system correctness; evidence E₁ might then be test results, and E₂ a description of how the tests were selected and the adequacy of their coverage, so that SC₁ is a subclaim that the system is adequately tested and argument step AS₂ provides justification for this. We need to be sure that the deployed software is the same as that tested, so E₃ might be version management data to confirm this and argument step AS₁ provides justification that the claim of software correctness follows if the software is adequately tested, and the tested software is the deployed software. Of course, a real assurance case will concern more than testing and even testing will require dozens of items of supporting evidence, so real assurance cases are huge.

Observe that the argument step AS₁ on the left of Figure 4 uses both evidence E₃ and a subclaim SC₁. By introducing additional subclaims where necessary, it is straightforward to convert arguments into *simple form* where each argument step is supported either by subclaims or by evidence, but not by a combination of the two. The mixed or free form argument on the left of Figure 4 is converted to simple form on the right by introducing a new subclaim SC_n and a new argument step ES_n above E₃.

The benefit of simple form is that argument steps are now of two kinds: those supported by subclaims

Here, **C** indicates a claim, **SC** a subclaim, and **E** evidence; **AS** indicates a generic argument step, **RS** a reasoning step, and **ES** an evidential step.

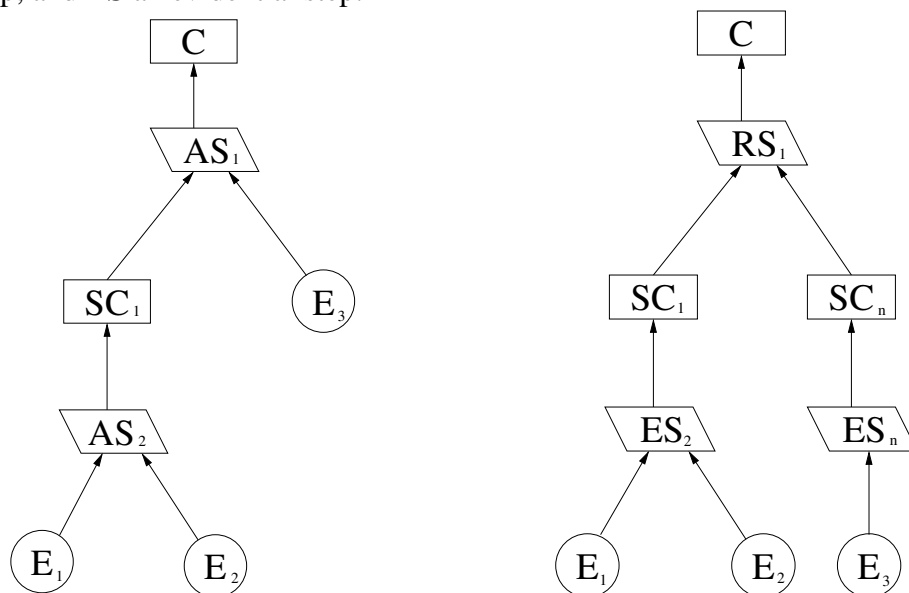


Figure 4: A Structured Argument in Free (left) and Simple Form (right)

are called *reasoning steps* (in the example, argument step **AS₁** is relabeled as reasoning step **RS₁**), while those supported by evidence are called *evidential steps* (in the example, these are the relabeled step **ES₂** and the new step **ES_n**) and the distinction is important because the two kinds of argument step are interpreted differently.

Specifically, evidential steps are interpreted “epistemically” while reasoning steps are interpreted “logically”. The idea is that evidential steps whose “weight of evidence” crosses some threshold are treated as premises in a conventional logical argument in which the reasoning steps are treated as axioms. This is a systematic version of “Natural Language Deductivism” (NLD) [62], which interprets informal arguments as attempts to create deductively valid arguments. NLD differs from deductive proof in formal mathematics and logic in that its premises are “reasonable or plausible” rather than certain, and hence its conclusions are likewise reasonable or plausible rather than certain [63]. Our requirement that the weight of each evidential step must cross some threshold systematizes what it means for the premises to be reasonable or plausible. Our treatment of reasoning steps shares with NLD the requirement that these should be deductively valid; this differs from other interpretations of informal argumentation, which adopt weaker criteria [25], and with most other treatments of assurance cases, which require only that argument steps are “inductively” valid (i.e., strongly suggest their conclusion, but are not required to entail it).

Since our treatment is close to formal logic, we adopt its terminology and say that a structured argument is *valid* if it is logically so and that it is *sound* if, in addition, its evidential steps all cross their thresholds for credibility. We have proposed Bayesian methods for evaluating the “weight” of

evidential steps [111] and recommend their use in “what if” experiments to hone judgment, but do not require their use in individual cases. We say that an assurance case is *justified* if its argument is sound. We believe that assurance cases should be subjected to *dialectical challenge* to evaluate soundness and infeasibility and overcome confirmation bias [112].

We suggest that conventional elements of IA systems, and even some of the innovative functions, can be assured by relatively straightforward application of assurance case methods. This would be assisted if the next iterations of guidelines such as DO-178C [104] and ARP 4754A [127] are cast as assurance case templates.

Here, we focus on more challenging aspects, including those that employ nondeterministic implementation methods such as search and opaque methods such as machine learning. As described in our requirements report [8], we suggest that runtime monitoring may be the most effective and credible method for assurance of such systems. We propose is not monitoring of individual functions (although that could be an effective assurance strategy at the function and component level), but monitoring of the entire process of flight. As well as providing an effective method for assuring IA systems [109], such high-level monitoring is an IA function in its own right, as inadequate monitoring and challenging are implicated in the majority of aircraft accidents attributed to crew error [53].

The basis of full flight monitoring is a model of “safe flight” and the top-level claim of the assurance case is that the system is safe, assuming the “safe flight” model is satisfied. Runtime checking for satisfaction of the model is relatively straightforward,⁵ so the bulk of the case should focus on correctness and completeness of the model.

We propose that the “safe flight” model should be decomposed into separate components that operate in parallel, coordinated by global variables such as “phase of flight” (takeoff, cruise, taxi etc.) or “nonstandard procedure” (engine-out, broken weather radar etc.). The claim for each submodel is that no (or acceptably few) good flights should violate the model (this is correctness or validity), but that all departures from standard or safe operation violate one or more models (this is completeness or infeasibility). The top-level of the assurance case thus reflects the structure of the model and we believe this can be iterated through all levels of the case and the model.

⁵This may be compared with pre-runtime checking of an implementation against a comparable model by model checking or theorem proving: Webster et al [137] describe model-checking an implementation against the British rules of flight.

4 Safety Relevant Requirements in IA Systems

To ensure that all safety requirements are generated when contemplating the transition from two pilot operations to an IA system that reduces, changes or eliminates a human's role, it is helpful to use a framework for organizing one's efforts.

The Honeywell team has considered the principles of crew resource management (CRM) between two human pilots to define the new tasks that automated systems need to accomplish and how these systems should interact with the pilot. A single pilot crewmate will require the next-generation IA system to communicate and collaborate as seamlessly and constructively as a two-pilot crew. Thus the most critical challenges in designing an IA system will be in communicating effectively, providing an efficient level of shared situation awareness, supporting state and mode awareness, and fostering acceptance and trust (while minimizing the negative effects of complacency).

For that reason, Honeywell has created a set of CRM-based requirements for IA systems' information acquisition, processing and interaction with the pilot. This set of requirements focuses on communication of intentions and planning, transparency in actions and decision making, error checking, conflict resolution, critiquing and notification of unexpected events, maintaining shared awareness, workload management and task allocation, distraction avoidance, and reducing nonessential alerts and communication during periods of high workload.

4.1 Operational Scenarios

The following operational scenarios were considered in this effort:

- Cruise; straight and level flight at FL350 heading 180° en route from SEA to PHX
 - ATC radio communication: Delta five two seven, Traffic two o'clock westbound
 - * Action: respond to ATC communication
 - * Action: visually locate traffic
 - * Action: call traffic in sight
 - * Action: monitor traffic
 - * Action: call traffic no factor
 - System fault: Pitot-static system (clogged tube)
 - * Action: monitor instruments; discover fault
 - * Action: perform checklist
 - System restored
 - * Action: observe clearance of error/fault condition
 - Weather phenomenon: icing
 - * Action: follow SOP
 - * Action: call ATC
 - * Action: complete checklist
 - * Action: request descent from ATC

- * Action: visual check for icing
- * Action: check system/instruments to indicate clear of ice
- * Action: call ATC and indicate condition
- * Action: return to flight path or deviate per ATC instruction Approach; descent through 3000 ft MSL, heading 258° ; conduct landing checklist; cleared for PHX RNAV Rwy 25L, 5 miles from FIXAR
- ATC radio communication:
 - * Action: respond to ATC clearance
- Preparation for approach
 - * Action: perform checklist
 - * Action: Check navigational charts
 - * Action: Check approach plate
 - * Action: Observe any NOTAMS
 - * Action: Get latest ATIS for PHX
- Descend to reach FIXAR at 7000 ft
 - * Action: Call reaching FIXAR, check speed and altitude
- ATC runway change - cleared for rwy 07L
 - * Action: respond to ATC clearance
 - * Action: Access and check approach plate for 07L
 - * Action: Set new approach in FMS
- Fly the approach (normal procedure). Emergency landing; Medical emergency: co-pilot is incapacitated, no alternate airport within 25 miles
- Prepare for emergency landing
 - * Action: Call ATC, indicate intentions
 - * Action: Set emergency transponder code
 - * Action: Check navigational charts and terrain display
 - * Action: establish immediate descent
- Find landing site
 - * Action: visually scan the area
 - * Action: Check navigation display
 - * Action: Identify landing site
 - * Action: Call ATC, indicate intentions
 - * Action: Set flaps and deploy landing gear
- Prepare for soft field landing
 - * Action: Check references as appropriate
 - * Action: Perform mental calculations for aircraft weight
- Perform landing
- Contact ATC with location, condition of crewmate

These scenarios were chosen to represent nominal piloting tasks including: flight operations, manual maneuvering, autopilot/autothrottle control, communication with ATC, navigation tasks, checklist tasks and charted procedures. Varying levels of flight automation and manual control were considered, including single pilot-only operations, single pilot with “super dispatcher”⁶ and distributed crew operations with a dedicated First Officer on the ground.

4.1.1 Critical Events

Each of these scenarios has been analyzed further with respect to off-nominal, unexpected or “surprise” events which will support the development of requirements for automation processing and interaction under these special circumstances. Events which have been considered include:

- Procedural change due to weather
- Runway change
- Automation subsystem failure
- Sensor failure
- System fault

4.2 Cognitive Work Analysis

The purpose of IA flight control systems is to reduce, change or eliminate human pilot roles and responsibilities. To discuss the implications of introducing an IA system, we must first study the current system (two human pilots) and identify the roles and responsibilities, operational scenarios, communications, exist-ing automation and information needs required for this system that are currently in place. Capturing this complex sociotechnical work system is not an easy task. There are numerous operations (some of which are not documented) that take place in a typical cockpit depending upon the operational scenario. Capturing each and every task is beyond the scope of this project, we instead captured the tasks for three crucial phases of the flight: cruise, approach, and emergency landing.

We considered a scenario where there are two pilots in a flight—a pilot flying (PF) and a pilot monitoring (PM). The PF is in charge of flying while the PM monitors the plane’s instruments, displays, communicates with Air Traffic Control (ATC), and assists the PF. The roles are interchangeable, so both pilots must be competent enough to handle all tasks. Further, some tasks may be performed by both pilots or shared (S) between them. Some tasks are done throughout the course of the flight, whereas others are done at frequent intervals or when needed. Tasks that are not regularly done are often triggered by an event, such as directions from the ATC or copilot, alerts from the control displays or the pilots’ sense that something is wrong. We found that tasks are often clearly defined.

⁶ A “super dispatcher” [36] is a ground-based pilot responsible for monitoring a number of active aircraft at one time, such that when an aircraft enters an “off-nominal” state due to an issue or anomaly, the pilot becomes a ground-based first officer dedicated to that aircraft).

There are certain sets of inputs required to perform the tasks. Pilots are often able to clearly describe the actions that are taken and the outputs that are expected on completion of the task. We also found that, a number of tasks are common to several phases of the flight. For instance, irrespective of the flight phase, the pilots monitor the instruments continuously. Also, several tasks are interdependent in the sense that, one follows the other such as sensing a hazard and taking mitigation steps.

To capture all these in a conceptually clean manner, we chose the *Cognitive Work Analysis* (CWA) approach—a structured framework specifically developed for considering the development and analysis of complex automated and semiautomated systems [76]. Unlike many human factors approaches, CWA does not focus on how human-system interaction should proceed, or how it currently works. Instead, “through a focus on constraints, it develops a model of how work can be conducted within a given work domain, without explicitly identifying specific sequences of actions.”

A consulting pilot who is experienced with dual-pilot, corporate jet operations was interviewed to support the CWA. A cognitive walkthrough was conducted in which we listed the PF, PM and “shared” functions for each scenario, creating a matrix of consecutive functions to information and communications required to perform each function in various environmental contexts.

First, a quick look analysis was performed to acclimate the pilot to thinking critically about the functions required to complete each phase of flight, as shown in Figure 5, without defining specific roles, activities or modalities of interaction. The activities recorded are listed in Table 1.



Figure 5: Phases Of A Flight (*Image courtesy of Federal Aviation Administration 4.1 Taxi*)

Next, a more in-depth work analysis was conducted for each of the three aforementioned scenarios. Each of the lowest-level functions was assessed in terms of logical trigger, inputs and outputs, and specific behavior currently required to perform each function. The complete CWA worksheet is delivered as a supporting document along with this report.

Flight Phases					
Pre-Flight Planning	Push back	Taxi	Takeoff/ Climb	Cruise	Approach/ Landing
Review mission and estimated departure time; Start APU	Coordinate with crew on safety briefing	Confirm taxi clearance	Select headings	Set radio frequencies	Get clearance
Select route; Get ATC clearances	taxi to movement area	Proceed to hold point	Select flaps	Follow charts for flight path	Execute STAR
Calculate flight duration; Load flight plan in FMS	Proceed to special services (de-ice) if req'd	Hold per controller instructions	Check speed	Engage autopilot	Apply speed brakes as req'd
Check route for terrain, restrictions; Check and confirm instruments	Check brakes	Enter taxi queue	Check winds	Review weather radar	Lower gear
Review weather and winds forecasts; Check and confirm panel settings	Check mode control panel, CDU	throttle up	Throttle back	Review traffic and terrain displays	Select flaps
Review NOTAMS	Confirm clearances, taxi instructions; Get situational awareness	check flaps, trim	Confirm operation as expected	Monitor winds	Check speed
Evaluate tech stops – places to get fuel; Sign maintenance logs	Crew Briefing – normal events, abnormal events, initial route and altitude briefing	Run through taxi checklist	Execute SID	Listen for ATC instructions/ advisory	Confirm clearance
Check fuel prices for destination and fuel stops; Start engines	Look outside	Get clearance	Raise gear	Adjust headings and altitude as required	Engage required Nav aids
Identify and review SID; Check ailerons, elevators, rudder	Look for traffic	Turn to assigned runway	Check flaps	Adjust cost index as required	Locate runway center line
Identify and review STAR; Confirm doors armed	Monitor party line		Adjust altitude and headings as req'd	Respond to equipment issues as required	Confirm ability to execute
Review pertinent charts; Turn on landing lights	Construct mental picture of traffic		Get wake turbulence situational awareness	Monitor instruments	Follow missed approach if req'd

Table 1: Cognitive Work Analysis Quick Look

Flight Phases					
Pre-Flight Planning	Push back	Taxi	Takeoff/ Climb	Cruise	Approach/ Landing
Determine fuel requirements; Complete pre-flight checklists			Make a last minute check of wind – position the controls for the aircraft type	Look up power setting based on altitude and temperature	touch down
Confirm seat configuration for weight and balance; Preflight aircraft				Convert TAS to IAS, and compare iPad Pilatus app to Honeywell FMW to confirm performance	slow aircraft
Calculate takeoff distance/ landing roll; Confirm databases are up to date				Confirm taxi clearance	
Take Weight and Balance, review limitations; Evaluate weather, route, and options					
Review aircraft speed limitations; Evaluate clearance vs. filed route, and update plan					
Document crew roster; Discuss desired route amendments					
Note FMS and EGPWS s/w configs; Verify cabin ready (varies for corporate vs. airline)					
Note VOR Nav configurations; Verify manifest and passenger documents					

Table 2: Cognitive Work Analysis Quick Look. Contd.

4.3 Stakeholder-Level Requirements for an IA System in CRM Context

Requirements exist at multiple levels in the process of system conception, development, and certification. The process starts with the conceptualization of the purpose and attributes of the system among the stakeholders including aircraft manufacturers, automation/cockpit suppliers, airlines, pilots, and regulatory authorities. Stakeholder-level requirements typically include gross functional performance, behaviors, and attributes of the system. This section explores the stakeholder-level requirements for an IA system that reduces, changes or eliminates a human's role in two-pilot operations.

The process of Requirements Analysis (RA) involves applying the combined results of user input, operational expectations and work analysis to the definition of quantifiable, relevant and detailed operational, functional and technical requirements. We have derived information requirements, functional requirements and communication requirements pertaining to the flight tasks and they are elaborated in the Appendix B. Note that we will not derive technical requirements or specifications, as we expect there will be multiple or novel technical solutions available to address the operational and functional interface requirements. Assumptions about the basic technical capabilities of the system are captured below.

4.3.1 Technical Assumptions

The CWA and RA we've conducted assume the system has certain technical abilities; as stated earlier, we will not detail the type, design or modality of the technical aspects of the system in order to ensure the design assurance applies to a variety of solutions and architectures. These high-level assumptions are enumerated below:

- Display software capable of real-time processing (latency, salience)
- Control software capable of real-time processing (latency)
- Integrated sensor system
- Decision-making logic
- Alerting mechanism(s)
- Voice recognition
- Either of the following, as required to support ATC communications:
 - Language model and syntax database, OR
 - Digital downlink
- Digital/analog input recognition, as required to support the pilot interface

4.4 Property-Based Specification of IA System-Level Requirements

At the next level are the system-level requirements and system architecture definition. The system design process establishes the boundaries and the context of system and further decomposes the system and into major subsystems and their interfaces. These include the IA system, on-board avionic subsystems, human pilot, cockpit displays, instruments, additional devices, and external actors such as ground stations and air traffic control (ATC). Absent such specifications about the system, one cannot rigorously discuss the implications of introducing an IA system in the control loop. For instance, without knowing how the pilot is going to communicate with the IA system, one cannot explore the advantages, failure modes and assurance strategies to guarantee smooth communication between the pilot and the IA system. System requirements drive the downstream activities of the development and verification of hardware/software components.

In this section we identify exemplars of system-level requirements that will satisfy stakeholder requirements. We first present a property-based requirements specification approach that allows the capture of key functional properties and attributes of the IA system especially from the perspective of design assurance. Next, we define examples of property-based specifications across the spectrum of the different categories of IA system functions. It is noted that the scope of this report is to identify examples across the functional spectrum to properly inform the assurance and safety considerations.

4.4.1 Property-Based Requirements

A well-defined set of requirements is considered to be a necessary prerequisite for efficient and effective communication between the users, requirements engineer, and the designer/constructor of the system. Most system requirements are often written in natural language since it can be easily written, read and understood by variety of stakeholders such as pilots, engineers, developers and certification agencies. However, despite all efforts they are vague, interpretable and dependent on the subjectivity of the author as well as the reader [37,93]. To improve this situation, requirements must be objectified, i.e. rendered precisely and be understood in the same way by all stakeholders concerned [93,95]. A variety of notations and languages to capture requirements, in both formal and informal means, were developed to achieve the desired level of precision and specificity.

Pure formal notations such as linear temporal logic are highly precise and are amenable to rigorous formal analysis and assurance. But they lack the ease and understandability of natural language and, more importantly, the requirement authors have to be highly skilled to write, and validate the requirements. To partly address this challenge, recent work such as ModelWorks [88] and Requirements Specification Language (RSL) [55] have attempted to capture requirements in formal notation, but with a slight natural language flavor. They also provide set of tools to perform a variety of automated analysis on the requirements. However, they still require some level of mathematical and formal logic skill to employ them.

To leverage the advantages of both natural language and formal notation, we propose a *property based requirements notation* that uses a controlled, structured natural language to specify behaviors of a system in an intuitively understandable yet unambiguous manner. Such a structured notation is amenable to machine interpretation and thus enables the transparent use of back-end automation tools to generate formal specification to enable downstream V& V and assurance activities.

Property based requirements (PBR) method is a way to specify requirements as a set of properties of system objects in either structured language or formal notations. The idea of PBR was first proposed by Dr. Patrice Micouin [94]. PBR exist within a model-based system design methodology, termed the Propriety Model Method (PMM) [95]. PBR are always passive, observable statements on system properties; a well-formed requirement is “a constraint applied to a property of the system when a condition or event occurs or a state is achieved”.

This notation focuses on capturing the properties of the system. System properties can either be specified as a combination of relational, mathematical and/or logical operations over objects of the system (constraints), or as a causal relationship between desired system behaviors and its causes (triggers and preconditions). This notation provides keywords such as ‘While’ and ‘Where’ to allow users distinguish between requirements about a specific state of the system or with a specific configuration (or enabled feature) respectively. Similarly, keywords such as ‘When’ and ‘If’ allow users to distinguish normal operational requirements from abnormal situation requirements. Further, this notation provides a set of relational, mathematical, logical and intent based constructs in a natural language form that helps standardize the way the system behaviors and causes are captured, yet be intuitive to the users.

PBR serve the role of clearly delineating the context and behaviors of the IA system in terms of well-defined system objects and their properties. This is crucial in our context since the tasks performed by pilots draw from accumulated learning from multiple source: standard operating procedure (SOP), formal education and training, total flight hours, knowledge of operating the aircraft, experience handling situations at various levels of emergency, and general common sense and intuition. Thus, the requirements for many of a pilot’s tasks and behaviors are implicit, centered on the pilot’s “model of the world”. When specifying requirements for the IA system, one needs to explicitly define the machine’s “model of the world” and specify properties that are grounded to specific entities in that model. Within the scope of this report, we do not attempt to define this model but provide examples of properties that would be specified.

In addition to facilitating requirements specification in the manner stated above, PBR enable a flexible and rigorous approach to design assurance and safety analysis. Verification and validation can be performed for certain properties at design time and for certain other properties using run time monitoring techniques. Notion of independent properties that are strongly tied to system objects lend similar flexibility in the definition of assurance cases. Section 3 contains more details on the assurance considerations.

4.5 Requirements Categorization

When defining requirements for an increasingly autonomous (IA) system to perform one of the pilots' tasks, a number of considerations come into play. Aspects that are implicit and natural for humans such as communication have to be explicitly designed into the automation. For instance, discrepancies between human judgment and an IA system's reasoning have to be minimal and resolvable. Automation surprises and confusions should be avoided. To do so, when defining the requirements for the IA system one has to account for the differences between a human vs. automation performing the tasks in both normal and abnormal operational scenarios.

To that end, instead of discussing individual task we identified 12 board categories to group the tasks based on the capabilities required to perform the task, the triggering event (if any), the communication required, and the criticality of the task to the safety of the flight. Below we explain each category by defining a few representative requirements expressed using the property based notation and examine their impact to overall system safety.

4.5.1 Instrument Monitoring

When the flight is in the air, both pilots monitor the instruments in the cockpit such as altimeter, air speed indicator, magnetic compass, heading Indicator, etc. at frequent, regular intervals. Throughout the instrument scan, the pilots mentally integrate the information presented on various displays with that obtained through their physical senses. When there is a mismatch between displayed information, or between the instrument and their visual or vestibular perception, pilots will focus their attention on troubleshooting the issue. The benefit of having a two-pilot crew, in this respect, is the ability to confer about the sensed information and the displayed information, whilst maintaining the primary task load (while the PF maintains flight control and the PM communicates and conducts appropriate checklists). Monitoring and integrating the data from instruments and displays in the cockpit, particularly for anomalies, is crucial since the results can determine the next time-critical course of action. Despite this importance, failure to properly monitor and cross check flight instruments has led to several accidents.

Consider the task of monitoring the altimeter to check the altitude. Typically it is monitored continuously from takeoff to landing. For instance, when the aircraft is in the descend mode, the altimeter reading is expected to decrease at the desired rate and be consistent with vertical speed indicator display. In addition to such normal checks, a number of safety concerns also come into play such as minimum altitude when flying over crowded cities and high raise buildings. One of the General Operating and Flight Rules laid out by FAA specifies that

“Over any congested area of a city, town, or settlement, or over any open air assembly of persons, an altitude of 1,000 feet above the highest obstacle within a horizontal radius of 2,000 feet of the aircraft” [2].

When capturing functional requirements for an IA system to meet the above safety constraint one has to take into consideration the role of the IA system and the flight status. While such requirements are crucial in normal operating conditions, they may have to be overridden in abnormal/emergency situations. In a PF role, under normal conditions, the IA system shall consider this constraint when setting the slope of descent (different task category). Whereas the requirement for an IA system in a PM role is,

While the role of IA system is PM, when the aircraft is flying over a crowded locale and the altimeter reading is equal to or less than 1000' (above the height of the tallest obstacle within a horizontal radius of 2,000' of the aircraft, then the flight monitoring system shall register 'below MSA' and shall present an associated alert to the crew.

The challenge here is that rules, dependencies and relationships between instruments and flight situation that are “known” to the pilot have not been documented in a systematic manner to capture them as requirements. In some cases, it is necessary for the pilot to use experiential knowledge coupled with instrument information in order to diagnose an environmental or mechanical problem well before an alert situation is present. For example, a steady decline in airspeed could be a sign of icing on the wings. The level of decline necessary to take action often depends on the type and size of the aircraft – so an experienced pilot of that aircraft type would be more likely to pick up on the condition faster than a novice. While the task requirements we have elicited from the pilots and documented in B is a representative subset, we believe that completely capturing the requirements for a real IA system is a much elaborate, involved task.

Further, differences between how a human pilot and IA system accesses the instruments has to be considered while capturing requirements. While the pilot may “see” the instruments, the IA system may have direct, seamless access to the instruments. On one hand, this is advantageous since display errors, precision issues, limitations in number of messages that are displayed, oversight and distractions are avoided. Conversely, this also means that the IA system could be receiving different data than what the other pilot sees leading to inconsistency and confusion between the pilot and IA system. Hence, this warrants additional requirements and well-defined process to resolve the discrepancy in the crew communication category to ensure that the pilots are in agreement. In the future, one could train the IA system to learn from the discrepancies and adapt automatically.

4.5.2 System Monitoring

In addition to monitoring the instruments, the pilots also monitor data from various analysis displays such as primary flight displays, flight management system displays, weather data and synoptic displays that assimilate the instrument data. The intent of monitoring these displays is to cross check the instruments and to identify anomalous situations.

Often pilots mentally merge the data from these displays and the instruments to come to a conclusion about the status and determine the next course of action. System diagnostics will often complement

a diagnosis made through visual or auditory analysis. For example, in propeller aircraft, carburetor icing can be detected by a drop in engine RPMs and the engine “running rough” (heavy noise and vibration). Multiple system indications and perceptual cues may be considered in diagnosing a serious fault or failure. For instance, observing a drop in fuel flow in addition to anomalous readings for fan speed and exhaust gas, and/or loud noises from the engine compartment may help the pilot make an early diagnosis of critical engine damage.

For instance, consider the task of determining a missed approach – a situation when it is judged that an approach cannot be continued to a successful landing. Typically, the conditions to determine a missed approach include a combination of several instruments, displays, ATC inputs etc. Capturing those conditions in requirements and precisely specifying their dependencies is crucial. A typical requirement for an IA system to determine missed approach is:

If one of the following conditions are true,

- *the approach is ‘un-stabilized’;*
- *the runway is obstructed*
- *Landing clearance is not received or is cancelled*
- *the altitude is less than or equal to the Decision Altitude/Height and the visual references are not established*
- *the aircraft position for controlled touch down within the designated runway touch-down zone is ‘Unsafe’*
- *ATC approved go-around during training.*

then missed approach shall be true, otherwise missed approach shall be false.

Conditions within the above requirement, such as “approach un-stabilized” are again decisions taken based on complex combinations of situation assessment, system monitors, etc. A related requirement to determine the stability of an approach:

When all of the following conditions are true,

- *the aircraft configuration is ‘landing’*
- *VREF \leq airspeed \leq VREF + 20kts*
- *Sink rate is \leq 1000 feet/minute*

then approach shall be ‘stable’, otherwise the approach shall be ‘un-stabilized’

For an IA system to make these decisions, in addition to the challenges mentioned for instrument monitoring, knowledge of how to safely deal with anomalous situations and knowledge about safe deviations to operating procedures has to be built in. In these scenarios, the communication between the IA system, ATC and other crew mates plays a crucial role. The IA system shall be capable of explaining its rationale to the crew member by identifying the root cause as well as determine if the explanations provided by the pilot is acceptable and will not compromise safety.

4.5.3 Environment Monitoring

In addition to monitoring the displays and instruments within the cockpit, pilots also visually look outside for traffic, obstacles in runway, weather and terrain conditions. In that respect, the IA system needs to “see”, using technology such as cameras, radars and computer vision, interpret what it sees correctly and make decisions based on that.

In aviation, visual meteorological conditions (VMC) are a set of conditions that determine if pilots have sufficient visibility to fly the aircraft maintaining visual separation from terrain and other aircraft. If the VMC is met, the pilots may choose to follow Visual flight rules (VFR), a set of regulations under which a pilot operates an aircraft in weather conditions generally clear enough to allow the pilot to see to visually avoiding obstructions and other aircraft. If VMC is not met, the aircraft is in instrument meteorological conditions (IMC), and the pilots shall obtain approval for and use the rules laid out in Instrument Flight Rules (IFR), in which the operation of the aircraft will primarily be through referencing the instruments rather than visual reference.

While the role of IA system is PF, when IMC is true, then the applicable flight rules shall be ‘Instrument Flight Rules (IFR)’ and the aircraft must be cleared for IFR flight.

Further,

When the course of a VFR flight enters IMC, and the aircraft has been cleared for ‘Visual Flight Rules (VFR)’, the IA must request clearance to amend the flight plan and continue IFR.

Further,

When the course of flight transitions into IMC, and the aircraft has not been cleared for IFR flight, or if the aircraft is not properly equipped for IFR flight, then the IA must request clearance to navigate away from IMC conditions.

When the course of flight transitions into IMC, and the aircraft has not been cleared for IFR flight, or if the aircraft is not properly equipped for IFR flight, then the IA must request clearance to navigate away from IMC conditions. To determine whether the VMC is met or not, as laid out in flight rules for different airspaces, the pilot should be able to clearly see a certain distance, recognize clouds and estimate the distance of the aircraft from the clouds. For an IA system, this requires a variety of sensors around the aircraft and sensor fusion algorithms to precisely estimate the distance. For instance, a snippet of the flight rules table for VMC is shown in 6.

The tabular form of VMC requirements, shown in 6, can be expanded to textual form as follows:

When the role of IA system is PF and the airspace is ‘Class C’ and the flight visibility is greater than or equal to 3 statute miles and distance from clouds below aircraft is 500 feet and distance from clouds above aircraft is 1000 feet and distance from clouds horizontally is 2000 feet then VMC is true otherwise VMC is false.

Airspace	Flight visibility	Distance from clouds
Class A	Not Applicable	Not Applicable.
Class B	3 statute miles	Clear of Clouds.
Class C	3 statute miles	500 feet below. 1,000 feet above. 2,000 feet horizontal.
Class D	3 statute miles	500 feet below. 1,000 feet above. 2,000 feet horizontal.

Figure 6: Visual Meteorological Conditions

Along the same lines, consider a task where the PM looks out through the window to assess the weather conditions for possible icing. An IA system performing this task shall have similar situational awareness. “Perceiving” ice on the wing will adjust flight control inputs to maintain the safe operating envelope, and engage de-icing systems. In very cold climates, it is possible for there to be ice on the runway even if it is not visible. A human may exercise extra caution in such a situation based on their experience, so the IA system has to combine past data, knowledge of terrain and the view to determine, predict and take (or suggest) precautionary measures as well.

All monitoring of the changing flight environment, including weather, obstacles, restrictions, etc., relies on the continuous updating of data sources. When operational issues exist due to changing conditions, the IA must have access to the most current and relevant data at all times, allowing it to assess the situation, make safe predictions, and take appropriate action.

In some cases, the IA system may be better than humans in quickly performing a variety of checks to detect problems. Consider the accident caused by frozen pitot tubes [27]. While the pilots relied heavily on the flight instruments, an IA system could have analyzed trends in the data based on other factors to quickly recognize and act on more reliable data, while applying checklist steps to alleviate the icing issue. In fact, an IA system has to be built such that it crosschecks failure modes of each instrument with potential problems, and decides the appropriate mitigation strategy or suggests the best course of action.

4.5.4 Flight Controls and Navigation

One of the primary reasons for monitoring is to infer the flying status and adjust the settings needed for flight appropriately. This category includes all such tasks that involve controlling and navigating the flight. Typically the flight control tasks are performed by the PF, while navigation tasks may be performed by the PF, PM or by both pilots.

In order to program an IA system to perform these tasks, one has to identify all the rules and procedures that determine what values be changed and when. The crew does this by accounting for the dependencies and redundancies among readings, history of the displays, the trends in change, inputs

from the other pilot, known conditions, etc. Because the flying tasks fall on the PF, when the IA system assumes the role of a PF it shall be capable of setting all the system parameters and taking relevant actions as a human pilot such as deciding the automation mode, enabling and disabling the autopilot, deploy flaps, desired altitude setting, etc. Since most of these tasks involve electronic controls and settings, the IA system shall have seamless access and authority to modify them directly.

A concern with letting the IA system modify the values is the scope of “automation surprises” – situations in which the human pilot is unaware of changes performed by automation. To mitigate this issue, requirements ensure that automation communicates its actions unambiguously to the other pilot should be defined.

Consider the task of applying speed brakes, a type of flight control surface used on an aircraft to increase drag or increase the angle of approach, during landing. Typically, if the airplane is too high and/or too fast before landing, the PF may instruct the PM to use the speed brake to get rid of some of the excess energy the airplane carries. Under normal circumstances, the PM performs the actions and confirms to the PF that the action was taken.

When an IA system is considered to perform this task, the level of automation expected from an IA system has to be clearly determined. If the IA system is expected to be “proactive” in identifying the need for speed brake deployment, then it shall be able to apply the brakes after confirming intended actions to the PF and confirm after the action is taken. A requirement for such an IA system shall be:

While the role of IA system is PM, when the conditions to deploy speed brakes are true and the confirmation obtained from PF to deploy speed brakes is true, then speed brakes shall be deployed; otherwise the speed brakes shall be not be deployed.

On the other hand, if the IA is not proactive, i.e., it has to wait for the PF to give directions on deploying the speed brakes, then the requirement shall be:

While the role of IA system is PM, when the conditions to deploy speed brakes are true and message from PF is ‘deploy speed brakes’ , then speed brakes shall be deployed; otherwise the speed brakes shall be not be deployed.

To avoid the situation where the PF may give an incorrect direction, we want to build in safety into the IA system by ensuring that the speed brakes will not be deployed in inappropriate situations.

While the role of IA system is PM, if the conditions to deploy speed brakes are false and message from PF is ‘deploy speed brakes’ , then speed brakes shall be not be deployed.

When the IA system is the PF, then the IA system may be designed to apply the speed brakes as appropriate and just inform the other crew mate. Again, depending upon the type of advances of the IA system, the requirement may vary. In its simplest form, a typical requirement shall be:

While the role of IA system is PF, when the conditions to deploy speed brakes are true, then speed brakes shall be deployed and message to PM shall be ‘speed brakes have been deployed’ .

In all the above requirements, a crucial aspect is being able to communicate with the pilot in an unambiguous and timely manner. While we concentrate on the aspects of control in this sub-section, we elaborately explain the communication aspects in the crew communication category.

For most tasks related to flight controls, like the above, the requirements can be inferred from SOPs, CRMs and training materials. However, the expected behavior in abnormal/anomalous situations is not well defined. For instance, consider the broadly specified responsibility of the pilot in command as defined by the general operating and flight rules, “*In an in-flight emergency requiring immediate action, the pilot in command may deviate from any rule of this part to the extent required to meet that emergency*” . To define the requirements of an IA system to perform the PF role in emergency situations, one has to prioritize and enlist all the actions that are appropriate for every emergency situation. The challenge is that for humans this relates to their knowledge of operating the aircraft, experience handling situations at various levels of emergency, general common sense and intuition; whereas for IA system all these have to be factored in. Defining safe actions for all emergency situations may be a demanding task. One approach is to gather knowledge or train the IA system from such past reports on deviations and use that data to determine the best course of action.

4.5.5 Fly manually (Direct Control Input)

Sometimes during flight, the PF must turn off the autopilot and fly using “stick and rudder” control. In both normal and emergency situations, when the IA system takes the role of a PF, it shall be capable of controlling the flight manually, i.e., using control column inputs. Flying manually places a complex visual, physical and cognitive load on the human pilot. The IA system shall perform tasks such as aircraft maneuvers, sense the movement of the aircraft in response to control inputs, enter new control modes as appropriate, etc. When doing so, the IA system must announce its actions to the PM unambiguously. It has to understand and respond to the suggestions and commands given to it by the PM or ATC in response to its action. While a human pilot could expect and rely on some actions to be taken by a PM when flying manually, the IA system in a PF role is expected to perform the PM role in parallel, to account for extreme cases when the other crew mate is disabled. One of the advantages of having a fully capable IA system for both pilot roles is that the IA system doesn’t experience the same workload constraints and pressure that humans do when multitasking in a complex environment.

When flying manually, human comfort has to be taken into consideration. The notion of “jerk” that is felt when speeds and altitudes are suddenly changed or the unpleasantness of turbulence are aspects that humans handle naturally, avoiding them when possible but prioritizing safety when necessary. Similar logic has to be built into the IA system.

For instance, when a pilot flying manually is asked to turn to a location heading 90 degrees from where he/she is flying. As a rule of thumb to avoid stress on the aircraft as well as avoid making the passengers uncomfortable, a good pilot will lead the turn with bank, make sure it is a coordinated turn using rudder input, keep a consistent 30 degree angle until about 5 degrees before the desired

heading, and roll out (decrease bank angle to zero) slowly to the correct heading. An inexperienced pilot on the other hand may vary his bank between 15 and 60 degrees or start the turn right away at a 30 degree bank angle and overcompensate by banking 60+ degrees, stressing the aircraft and the passengers. To ensure that the IA is indeed a “good” pilot, flying related requirements have to be specified clearly.

In the following requirement:

While the role of IA system is PF, when the ATC commands immediate left turn, then the control stick is turned left at an optimal safe bank angle and sustained until the requested heading is reached and rotated back to straight and level flight on the requested heading.

The optimal safe bank angle depends upon the aircraft and is specified in the pilot reference handbook. To perform such maneuvers, the IA system shall be capable of referencing such information and acting accordingly.

4.5.6 Crew Communication

Communication among the crew is crucial for safe operation. In fact, crew resource management is an area of training in aviation that attempts to avoid human error that can have devastating effects on air safety. When IA systems are used in place of humans, seamless and unambiguous communication is of utmost importance. This is because poor or nonexistent communication between the existing automation/user-interface and crew has been reported as a significant factor in many accident investigations. Hence, one has to assess the various kinds of human-automation interaction that can take place in a cockpit.

Communications can be as simple as sharing and acknowledging information giving specific directions in both verbal and non-verbal form. For instance, pilots may physically point to a reading to indicate an anomaly or may nod the head as an acknowledgement. Further, since interactions are often multi-step, the context of the conversation has to be tracked by the IA system. For instance, pronouns may be used following a discussion, like “reset that” . While such interactions are natural to humans, the automation shall be built with the ability to precisely recognize the voice, accent and gestures by the crewmate. IA communications, alerts and responses should not cause undue annoyance or distraction to the human crew mate. Cockpit voice recordings of various air disasters tragically reveal first officers and flight engineers attempting to bring critical information to the captain’s attention in an indirect and ineffective way. By the time the captain understood what was being said, it was too late to avert the disaster. Hence, when designing an IA system, the way it communicates with the pilot – both when it is in the PM and PF role should be explicitly stated as well as elaborately simulated and tested.

Further, one of the common concerns with automation is “automation surprises”, i.e. situations where crews are surprised by actions taken (or not taken) by the automation. The IA system shall unam-

biguously communicate what it is doing to the pilot as well as explain the rationale and intent of its actions when queried by the crew mate, in a way that can be “understood” by the human. Hence, one of the aspects when defining requirements of an IA system is to obtain agreement before it performs any action that are not prompted from the crew mate; same way, if an action is called out by the pilot, the IA system shall either confirm after performing the action, or clearly explain why the action was not taken.

Let’s consider a concrete scenario of applying speed brakes, discussed in Flight Controls category. When we consider a “pro-active” IA system that prompts the PF before performing the action:

While the role of IA system is PM, when the conditions to deploy speed brakes are true, then message to PF shall be ‘Can I deploy the speed brakes’.

On the other hand, if the IA is not proactive, i.e., it has to wait for the PF to give directions on deploying the speed brakes, then the requirement of such a system shall be:

While the role of IA system is PM, when the conditions to deploy speed brakes are true and message from PF is ‘deploy speed brakes’ and speed brakes are deployed, then the message to PF shall be ‘speed brakes are deployed’.

In the event of a problem or erroneous situation, the IA system shall be capable of clearly communicating the action and the rationale for it.

While the role of IA system is PM, if the speed brakes have not been deployed due to hardware error, then message to PM shall be ‘speed brakes have not been deployed, since there is a hardware error with the speed brakes’.

When the IA system is prompted by the PF to deploy speed brakes, but the IA system decides that it is not the right time for the action since some necessary conditions are violated, say air speed is too high, then it shall communicate that clearly:

While the role of IA system is PM, when the conditions to deploy speed brakes are false and the message from PF is ‘deploy speed brakes’ , the message to PF shall be ‘The air speed is too high. Are you sure you wanted the speed brakes to be deployed? It could be hazardous’.

Whether the pilot can override such advice from the IA system and command it again to deploy the speed brakes depends upon the level of automation and authority for the IA system. In any case, the IA system shall also be capable of repeating the explanation, when asked by the PF. In situations where the rationale is not immediately communicated, when prompted for explanation from the PF, the IA system shall be capable of providing a reason. For instance:

While the role of IA system is PM, when the previous message to the pilot is ‘speed brakes cannot be deployed’ and query from pilot is ‘Why’ or ‘What’s wrong’ , then the message to the pilot is ‘aircraft has to be in approach mode for speed brake deployment’.

When the IA system is the PF, then the IA system may be designed to apply the speed brakes as

appropriate and just inform the other crew mate. Again, depending upon the type of advances of the IA system, the requirement may vary. In its simplest form, a typical requirement shall be:

While the role of IA system is PF, when the conditions to deploy speed brakes are true, then speed brakes shall be deployed and message to PM shall be ‘speed brakes have been deployed’.

These phrases and sentences used in the above requirements are just used to illustrate the possible communication patterns. However, when defining the requirements of a real IA system, the protocol of communication that typically occurs in the cockpit has to be adequately accounted, so that the message is understood and interpreted precisely by the pilots.

4.5.7 Communicate with ATC

Apart from the interaction among the crew, communication over the radio is also a crucial task of a pilot. To so do, the IA system shall be able to set navigational and radio frequencies. Currently, the pilots reference various sources of information (physical books, e-books and inputs from ground station) to decide the right frequency and enter it into the system. For an IA system to perform this operation, it shall have the ability to specify values to any settings that pilots can alter. It has to be up-to-date with the current reference manuals.

Tasks such as “Alert crewmate to radio call” will require the IA system to precisely process spoken words over the radio and inform the crewmate in a timely and appropriate manner. The IA system needs to understand call signs, standard commands and data exchanges in both directions. It needs to generate speech to supply information, send alerts and query ATC. The IA system will have to listen to communications between the pilot and ATC as well as able to communicate directly with ATC if the pilot is out of cockpit or incapacitated.

The IA system must be prepared with a pilot-ATC communications database containing standard FAA and EASA terminology, and appropriate pilot responses to ATC queries and commands. This directory shall be robust enough to consider variations of phrases, colloquialisms and abbreviations, and its speech recognition engine must be reliable enough to detect a range of human accents and dialects. An example of a communication requirement would be:

While the role of IA system is PM, when the aircraft ownship call sign is detected over the radio, then the IA system will respond to the subsequent communication string in accordance with the communications database.

Further,

While the role of IA system is PM, when an ATC command is received requiring PF action, and the PF begins to take the appropriate action, the IA will respond to ATC with an appropriate affirmative read-back in accordance with the communications database.

When instructions are longer, the IA system shall also be capable of performing recording/playback functions. Currently, when a long instruction is to be executed, one of the pilots writes them down for read back. With automation, this will now have to be a recording and then display or an audio replay feature. While human entry/read back errors are avoided using automation, both the pilots and ATC have to be trained to adapt to this change in the way the IA communicates.

Further, depending upon the level of automation, the IA system can be designed to remind the pilot if he/she has not responded to the call from the ATC.

While the role of IA system is PM, when an ATC command is received requiring PF action, and the PF does not take action within XX (time) of receiving the command, the IA system will repeat the command to the PF.

4.5.8 Check Reference Material

Pilots typically look for navigational aids, reference aircraft/airline/FAA regulations, charts and equipment manuals at various instances of time during flight. They carry hard or soft copies of the materials with them on board. When an IA system is used, it needs to be loaded with current versions of those materials and shall be able to perform independent lookups. For instance, looking up power setting based on altitude and temperature by referencing manuals, flight and weather displays in order to adjust power accordingly and acknowledge that setting is a common task of the crew. Since both pilots have the ability to do these tasks, in situations where the pilot actions are based on different version of references, the IA system shall be capable of identifying the discrepancy (based on the pilot inputs or actions) and communicate it to the pilot appropriately. Further, the IA system shall account for the human delays in manually referencing the values.

4.5.9 Calculations

Calculations are often an intermediary step between monitoring/referencing material reference and taking appropriate actions/communications based on those observations. For instance, calculations based on aircraft weight, knowledge of system status and expected maneuver are performed in order to appropriately input values to FMS. While performing such calculations, conversions based on charts and tables are also performed. Further, some computations may also involve conversions. For instance, most overseas airports give altimeter settings in hectopascals (hPa) (millibars). Therefore, it is imperative that the IA system is able to convert inches of mercury to hPa or hPa to inches of mercury for using in relevant computations. The precision of these calculations largely contribute to safe operations of the flight. For instance, the calculations can be as simple as:

*The Max Drift Angle shall be $(60/\text{Aircraft True Speed}) * \text{Wind Speed}$*

*The Crosswind speed shall be $\text{wind speed} * \sin(\alpha)$*

*The Headwind speed shall be wind speed * cos (α)*

where α is the angle of the wind from direction of travel.

4.5.10 Troubleshooting and Checklists Actions

Troubleshooting is a crucial capability of pilots. An IA system is expected to detect errors based on system display, instruments or computations, as well take corrective actions for the same. To do so, the IA system needs to have the latest versions of troubleshooting procedures as well as access to other systems to verify readings or change settings to perform the actions. The IA system shall also have capabilities that will verify the clearance of the fault and next perform post-error actions. Additionally, it needs to recognize pilot confirmation of actions taken or problems encountered on a step and be able to prompt the pilot regarding next or missed items.

Similarly, the IA system shall also be able look for a checklist at the appropriate time and follow it with/without having to be instructed from the PF. For instance, when the PF instructs the crew to prepare for landing, the IA system shall be capable of identifying the appropriate checklist and perform the actions as specified such as turning the landing lights on. Where necessary, the IA shall take corrective action to address issues completing the checklist, and the crew mate when a checklist is complete, or has failed to complete. If the IA system receives a higher task, it shall queue up the tasks and perform them in the order of their priority.

The IA system shall also be able to task without having to be instructed by the PF. For instance, when an emergency is encountered as indicated in the SOP, setting the emergency transponder code is a crucial task. It shall recognize emergency situations and follow SOP to enter the appropriate transponder code, without having to wait for explicit instructions from the PF. However, when doing so the clearly communicating the actions to the crewmate is crucial. An example of this type of requirement would be:

While the role of IA system is PM, when conditions indicate a possible fuel leak, then the message to the PF shall be 'fuel leak is suspected' and the FUEL LEAK CHECKLIST is accessed from the emergency procedures database and each action from the check list is performed and verified in order.

Further,

When an action from the FUEL LEAK CHECKLIST is executed and verified by the IA, then the message to the PF shall be 'the action <details of action> has been successfully performed and <number of actions> is remaining to be complete'.

Further, if there are any issues with completing the tasks, in addition to reporting that problem the IA system shall also be capable of assessing whether to proceed with the next item, abort the operation and start over again or ask for pilot/ATC inputs. An example for such a requirement is:

When an action from the FUEL LEAK CHECKLIST is not executed or verified by the IA, then the message to the PF shall be ‘the action <details of action> cannot be performed and the next action in the check-list will be performed’.

On the other hand, when the crew mate is performing checklist actions, the IA system shall be adaptive enough to recognize benign/malign intent of the crew, when an item is skipped, or when there is a change in the sequence of the checklist items. This is essential in emergency landing procedures, since the crew member might inadvertently change the sequence of operation. The IA shall always maintain awareness of the crew mate’s task load and queue, and indicate/suggest sharing the load when the crew mate’s workload is saturated.

4.5.11 Ground Maneuvering

After touchdown, the aircraft is slowed to taxi speed and exit the runway. For taxiing, the power is reduced to idle, brakes are applied as appropriate, flaps are retracted, and the route is continuously scanned for possible obstructions. For an IA system to perform these tasks, it has to steer, brake, control the engine throttles, continuously scan outside the aircraft, and communicate with the control tower.

Consider the general safety properties for taxiing is to “ensure that there is no danger of collision with any person or object in the immediate area by visual inspection of the area” and “Aircraft shall be taxied in accordance with the prescribed taxiing patterns established by the airport.” While we one can specify requirements for the IA system to sense the surrounding and reference the taxiing pattern to maneuver the aircraft, even when some of the data is not available the IA system shall perform safe maneuvers.

Several accidents during taxiing have been reported with root causes ranging from poor visibility, incorrect computations etc. For example, the incident that occurred in 2010, when an Antonov An124 aircraft was taxiing in to park in normal night visibility, it collided with two successive lighting towers on the apron. This damaged both towers and the left wingtip of the aircraft. The root cause is pilot could not discern the lateral clearance with obstacles.

To avoid this issue, the IA system has to have adequate data about the taxiing path in the airport, aircraft dimensions as well as be fitted with enhanced sensors to detect obstacles and other aircrafts. Since low visibility during snow has repeatedly been a cause for several incidents, the sensors fitted for the IA system should be robust to weather conditions. Safety requirements would typically be:

While the role of IA system is PF, where the aircraft is in taxiing mode, if the clearance between aircraft wings and obstacles on the sides of taxiing path is not available, then aircraft taxiing speed shall be less than 10km/hr. and the obstacle sensors in the wings shall be activated.

While the role of IA system is PF, when obstacle sensors in the wings is clear for more

than 10 feet from aircraft, then the aircraft moves forward in the taxiing path, otherwise the aircraft is brought to a halt and the ATC is notified

4.6 Addressing Communication and Trust

Recent advancements have been made in human-IA trust interaction, incorporating bi-directional communication between human pilots and machine partners in the form of cooperative decision-making assistants and situation awareness transparency models. Key characteristics of these digital assistants include transparency (purpose and goals of the IA; reasoning process; how decision elements are weighted), shared language elements (terms which are familiar to the human pilot) and negotiation (allowing the human to change weights and add elements of consideration). At another level, the IA should give the human cues as to potential future outcomes and performance history when decisions are being made. The IA should be evaluated for the quality, quantity, timeliness and effectiveness of information provided to the pilot to support an “effectively transparent” user interface.

Important features to test in a communication system include:

- Bi-directional dialogue
- Shared goals
- Communication
- Trust
- Shared Awareness
- Flexibility/Robustness/Resilience

When evaluating IA training materials and methods, we can extend our assurance method to evaluate “calibrated trust”, which is a term used to describe trust based on pilots’ experience with an automated partner’s capabilities and limitations. The most effective utilization of trust-enhancing technology can be realized when the human pilot has sufficient information and insight to correctly deploy and interrogate the conclusions of the automation system. The effectiveness of IA systems is limited by the machine’s current inability to explain their decisions and to act symbiotically with human users. In the current state, the best performing autonomous systems, whose models reach impressive prediction accuracies, have nested non-linear structures that make it hard to decipher what is going on inside the system; it is not clear what information in the input data makes them actually arrive at their decisions [140].

To address this issue, new research and technology development programs in ML and IA interaction have focused more heavily on the concept of “Explainable AI (XAI)” [65], where the logic and rationale supporting AI decisions is presented to pilots (Figure 7). New IA systems will explain their rationale, characterize their strengths and weaknesses, and convey an understanding of how they will behave in the future.

While autonomous systems based on Machine Learning inherently have the capability to offer full

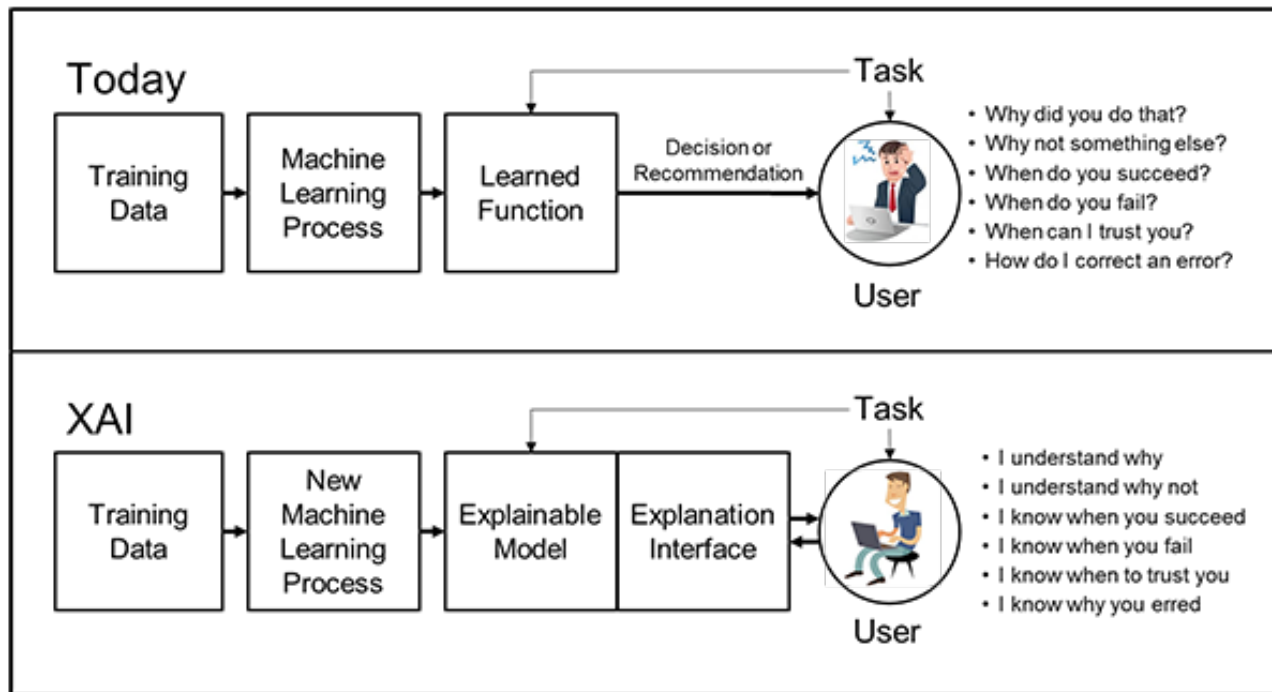


Figure 7: Explainable AI (XAI) Concept

transparency, i.e., output an explanation of their action, what is needed is a human-understandable justification that, unfortunately, is something entirely different than transparency. To explain, transparency is identifying what algorithm were used, parameters learned and math behind the choice of an action. This allows one to know the “physics” of the system. While interesting to developers and researchers, this information is of no use to end users who will collaborate with such systems to perform day to day tasks. On the other hand, the concept of justification involves communicating what the machine was thinking the same way a human would have thought, such as “Since I saw event X in the context of Y, I took action Z”. Justification tell us the rationale in a human understandable form. This justification is what we will refer to as “explainability”, a crucial requirement for an IA system to work on reduced crew operations.

Explainability is a scientifically fascinating and societally important topic that touches upon several topics of research. Some of them are:

Bias: When AI systems are incorporated into collaborative, critical environments such as reduced crew ops, issues of bias and discrimination can be serious. Given the difficulty of understanding how an AI produces a result, bias is often discovered after a system has been producing unfair results in the wild. Hence, the problem of ensuring that the AI hasn’t learned a biased view of the world based on shortcomings of the training data and approaches is an open challenge and there are currently limited/no strategies to address bias and fairness.

Level of Transparency: Striking a balance between the abstraction of the explanation, versus con-

veying a true understanding without hiding important details, is an open problem. While many support the view that transparency builds trust, in some settings, providing detailed information can lead to less efficiency. In fact, too little/too much information can increase the divergence between the mental models of humans and AI that can be counter-productive.

Security: Explainability poses a set of security concerns. Actors with misaligned interests can abuse information as a manipulation channel, inappropriately use it to override decisions. How can the system defend itself from malicious actors querying is a crucial question to be answered in collaborative safety-critical environments.

Language: The way an explanation is conveyed may differ with languages. The art of verbal/non-verbal communication that human pilots have mastered is, unfortunately, a tough problem for machines. Hence, being able to unambiguously understand a query, clarify without causing frustration and provide an explanation is another open area of research.

Besides being a gateway between AI and society, explainability is also a powerful tool for detecting flaws in the model and biases in the data, for verifying predictions, for improving models, and finally for gaining new insights into the problem at hand. Furthermore, explainability was presented as prerequisite for solving legal questions which are arising with the increased usage of AI systems, e.g., how to assign responsibility in case of system failure.

The State of the art strategy for achieving “explainable” systems combines sophisticated human-computer interface that translates the underlying models into understandable and useful explanation dialogues for the end user. However, since a key reason for building AI systems is to exceed human performance, a consequence of such superhuman performance may be that there is no explanation for how it works that is easily digestible and assurable by humans [125, 140]. For instance, the winning move of the AlphaGo system was commented on by a Go expert [92] in the following way:

It's not a human move. I've never seen a human play this move.

The goal for IA systems is to employ ML techniques that produce more explainable models, then translate these models into understandable and useful explanation dialogues for the end user. XAI should be used to establish pilot trust in automation and aid in shared decision-making. Providing reasoning for each automation decision will also enable the pilot to make informed decision about overruling automation if necessary. XAI is essential to pilots' understanding and trust of the IA system, but design for XAI should be balanced with performance efficiency. Future research is needed to define a range of design trade studies to develop efficient and effective XAI.

When effectively trained on an IA system, pilots should have a general understanding of what can and cannot be achieved using the automation. This will ensure the right utilization and prevent over- or under-trusting the automation. A required level of minimum training to safely and effectively interact with the IA system should be recommended for all pilots. This will ensure that pilots are aware of the basic limitations and capabilities of automation as well as their own roles and responsibilities within the system.

Irrespective of the level of automation, human-IA teaming should garner greater overall effectiveness, efficiency, reliability, simplicity, economy, and system safety rather than relying on automation or human performance alone. The human-automation teaming should improve overall performance and should not be implemented simply because the technology is available (technology-centered automation). The design should ensure that the human is involved in ongoing operations, and appropriately informed to maintain awareness of the situation and other status of automated functions. Behavior of automated functions should be predictable, offer the human an appropriate range of options, monitor human actions to minimize, resist, and tolerate errors, and be capable of being overridden by the human in an emergency. A clear indication of the current operating mode selected shall be provided to the human at all times. A fundamental mode is automation status (e.g. manual or partial or full automation). Pilots will not be expected to understand the ML/IA algorithms that are core to the systems, but they will be expected to trust these systems.

5 Safety Analysis Techniques

In the previous sections we summarized the design challenges that IA systems may introduce. The breadth and scope of such challenges are very large. Recent research [47] has already documented concerns with respect to the applicability of ARP4754A within the context and challenges associated with today’s highly integrated systems. The emerging challenges of deep-learning and artificial intelligence may further stress and surpass the effectiveness of existing certification guidance. However, new technologies and implementation approaches continue to be developed. Some of these, if applied correctly, may help mitigate the emerging risks. One such approach is STPA (Systems Theoretic Process Analysis) [85]. The STPA approach is based on the new accident causation model STAMP (System-Theoretic Accident Model and Process). It provides a good starting point for analyzing safety issues and hazards from a “system’s thinking” perspective. It may therefore compliment and augment some of today’s hazard analysis techniques that are largely component failure centric and based on reliability theory. Central to STAMP is systems theory. In other words, the system is modeled as socio-technical hierarchical control structures with components (controllers, actuators, sensors and controlled processes), control actions, and feedback loops. This allows for a common hazard based approach to be applied across the entire socio-technical environment. Therefore STPA is equally applicable to supporting the analysis of regulatory guidance and the effectiveness of safety management systems and operational procedures—down to the details of a particular subsystem design and implementation.

The hierarchical control structure of the system, derived from system functional requirements, serves as a basis for STPA. This guides the analyst to identify unsafe control actions leading to hazards and their causal scenarios. The causal scenarios identified using STPA help to identify and plan appropriate mitigation strategies. The advantage of STPA is that it views the overall system’s perspective and the interactions among its components when analyzing hazards.

In its original form, STPA states “what” steps should be done, and does not specify a meticulous approach to performing the steps. This makes STPA a largely manual process,⁷ whose success is predicated on the skill, domain knowledge and judgment of the person performing the analysis. Hence, when it comes to analyzing the hazards of newly conceptualized, futuristic systems with no history of usage or user experience, such as an increasingly autonomous (IA) controller for reduced crew operations in aircraft, it is unclear if STPA alone will be adequate to identify all of the hazards.

Recent work by Sou [132], however, has illustrated its effectiveness with respect to multi-system, interaction-related hazards. Sarcini [116] has also successfully applied it to the analyses of aircraft operational procedures. In addition France et al. [54] have recently extended the approach to take into account the human mental models in operational context to support the analyses of human-related causal factors that may yield unsafe control actions. Given the aforementioned attributes, we believe

⁷Although recent work by Thomas [133] has started to introduce limited automation support

that STPA and its existing extensions may provide a solid foundation for the analyses of IA reduced crew operations. That said, additional refinements and integration may be necessary to fully meet the challenges of these systems. For example, a closer integration of STPA and simulation based scenarios may be required and improved provisions for architectural refinement and analysis within the STPA framework may be necessary.

In the rest of this section, we describe our experiences applying STPA to an aircraft with an IA controller and investigate the benefits/gaps applying STAMP/STPA in that system.

5.1 Overview of STPA

Hazard analysis with STPA begins with an analyst identifying the kinds of accidents they wish to prevent and defining the hazardous states that could lead to those accidents. These definitions set the scope of the analysis. Then one also needs to build a model of the hierarchical control structure of the system, which shows how different entities within, and external to, the system interacts with it. This helps understand the roles and responsibilities of controllers at each level, as well as the control actions they perform, feedback they receive, and the process model needed to perform their tasks safely. Based on the hierarchical control structure, two main steps of the STPA analysis build upon these foundations to identify possible causes of accidents.

In STPA Step 1, the analyst identifies *unsafe control actions*—actions that could lead to a hazardous state by violating the system safety constraints. For each control action in the safety control structure, four types of unsafe control actions can be identified and typically presented in table format, with each of these four types in a separate column, and each control action in a separate row.

- A control action required for safety is not provided
- An unsafe control action (UCA) is provided that leads to a hazard
- A potentially safe control action provided too late, too early, or out of sequence
- A safe control action is stopped too soon or applied too long (for a continuous or non-discrete control action)

To partially formalize the specification of UCAs, Thomas [133] defines UCA to consist of four key components: (1) The *controller* is the entity that can provide the control action. (2) *Type* refers to which of the four types the action belongs to: provided, not provided, etc. (3) *Control action* refers to the action itself, or the link in the control structure that was affected. (4) *Context* describes the conditions under which the action leads to a hazard. It is also convention to identify the possible hazardous outcomes of each UCA by referencing the relevant hazards after the UCA statement. This provides traceability that can be carried throughout the analysis. Later, designers and engineers may want to mitigate the most serious hazards first, and can look to the UCAs and causal scenarios linked to those hazards.

In STPA Step 2, after UCAs have been identified, the analysts must identify scenarios that explain why each unsafe control action might occur, including ways in which control actions provided may not be carried out. The goal of these scenarios is to explain how, through a combination of factors, the unsafe control action may appear reasonable in context. Causal scenarios often involve the concept of process models, which are the internal representations that a controller uses to understand the controlled process. This model must include relationships among system variables, the current state of those variables, and ways in which the process can change state. This model enables the controller to interpret feedback to understand which actions are needed. While process model flaws are a common factor in many unsafe control actions, factors that contribute to hazardous states may occur anywhere in the control loop and could easily propagate to other parts. To write meaningful scenarios requires addressing how the system as a whole can experience dysfunctional interactions that lead to hazardous states. This model is meant only to provide guidance from which to begin brainstorming and developing causal scenarios, a process that ultimately relies on creative thinking and familiarity with how the system components interact.

5.2 Tooling

While there are a number of software tools that support hazard analysis with STPA, we choose to capture STPA using SAHRA (STPA based Hazard and Risk Analysis) tool [83]—developed by Safety-Critical Systems Research Lab of the Zurich University of Applied Sciences—for several reasons. It is an extension for the popular Sparx Systems Enterprise Architect UML/SysML modeling tool, which enables practitioners to adopt the STPA for their engineering and analysis processes. SAHRA improves the analysis workflow by not only supporting the complete STPA process but by offering a unique way of capturing STPA using the visual style of mind maps. It comes with a special Model Driven Generation profile which provides additional toolboxes, UML profiles, patterns and templates for STPA modeling. The extension provides a context-sensitive object browser and editor for STPA, diagram elements and special editors for performing STPA Step 1 and Step 2. It is also seamlessly integrated into the widely used Enterprise Architect (EA) that inherently supports the integration of STPA modeling within the larger context of SysML and UML system and software modeling refinements.⁸ EA also provides multi-user support with a security permission system, scripting and automation API, SQL searches, configuration management integration, report generation, and modeling functionality that enables the synergy between requirements, safety analysis and architecture. We believe that the integration of different model perspectives within an integrated modeling environment and context will introduce great benefits. However, we also recognize the challenges associated with the different abstraction levels that may exist within each modeling perspective.

⁸The ability to integrate these modeling languages, was the main driver for our choice of tool.

5.3 Landing Gear Deployment - Case Example

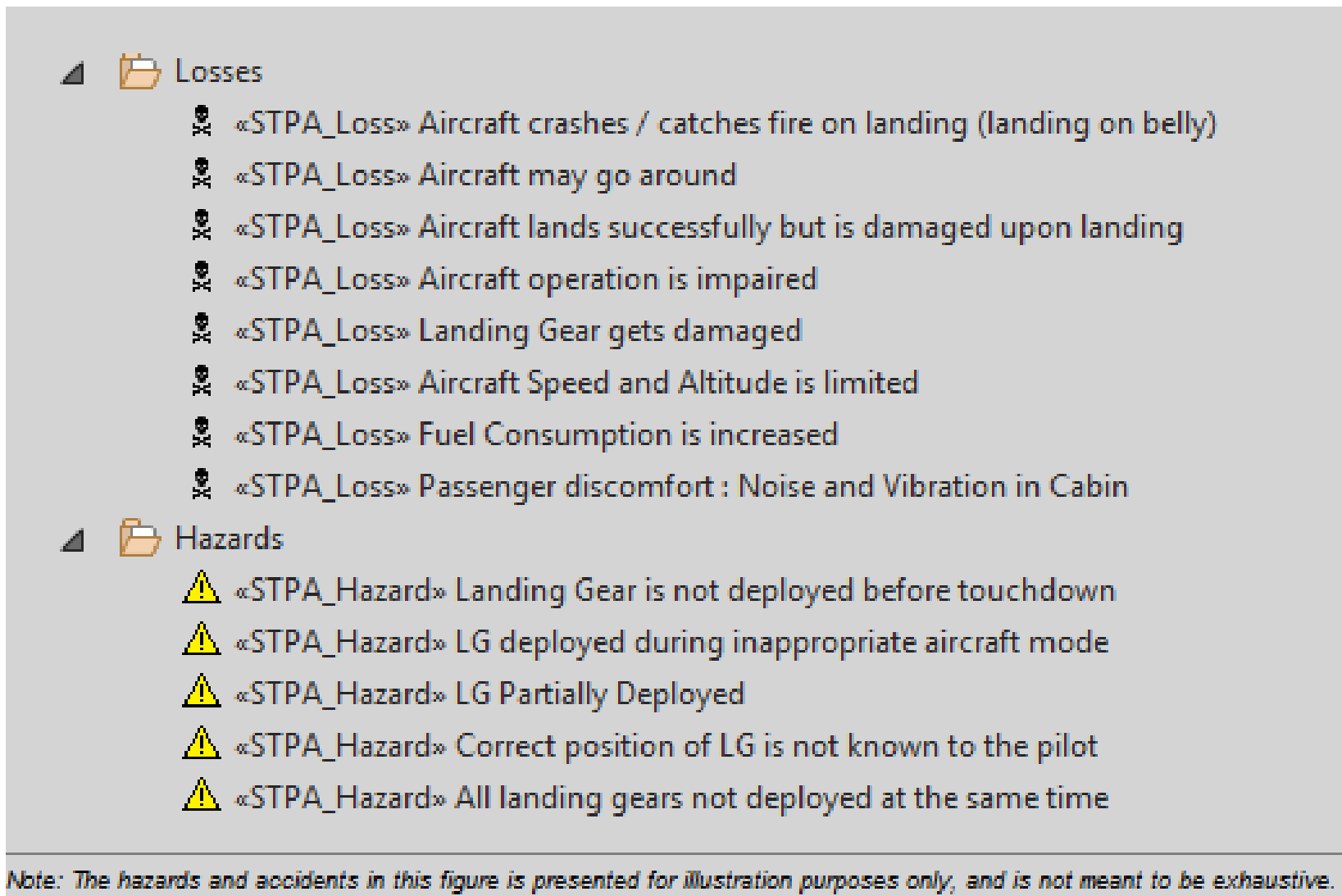
To explore the effectiveness of STPA technique in identifying and analysing the hazards of reduced crew operation by introducing an IA system in the cockpit, we applied it to the scenario of providing the landing gear deployment command when there is an intent to land the flight. Typically, in a two crew flight the landing gears are deployed by the non-flying pilot or monitoring pilot (PM) based on various factors such as aircraft status and mode, ATC clearances to land and instructions from the flying pilot. The PM checks the conditions suitable for landing gear deployment, deploys the landing gear as well as takes care of all necessary communications with ATC and co-pilots prior and post deployment. The STPA for the landing gear, explained in the report, was performed for demonstration and illustrative purposes only; the aim of this exercise was to understand the pros and cons of applying this technique to a reduced crew operation with an IA. Hence, it is neither exhaustive nor an exact representation of an actual STPA for such a system.

As a first step we captured the system fundamentals: the system accidents and system hazards. At this early phase, the system is the overall vehicle. The context and environmental assumptions are accounted at this stage when defining the hazards and accidents. Figure 8 lists the system accidents and system hazards defined for this case study. Notice that these are defined at a very high level and in fact they are similar to the flying with traditional crew scenarios. These accidents and hazards help define the scope of the effort, and all results from the analysis will be traced to these.

Unlike the usual way of capturing accidents that lead to catastrophic losses, we also captured those that caused degraded operation, annoyance as well as degraded safety margins that may have operational, financial and performance losses. For instance, an “aircraft go around” will lead to schedule delays that, in the long run may affect the airline reputation. Further, some of those concerns such as “aircraft altitude/speed is limited” is of importance for downstream developmental consideration. We annotated each accident with its impact (or severity) based on its context and associated the hazards that could potentially lead to it. Since a hazard can lead to many accidents, many hazards can lead to an accident, there is a many-many relationship among the hazards and accidents. For instance, the hazard “*Landing gear not deployed before touch down*” can lead to “*Aircraft crashes/ catches fire on landing*”, whose impact is catastrophic, when the pilots are not aware of the landing gear status. The same hazard can result in “*Aircraft may go-around*”, resulting in major schedule and cost impact, when pilots have recognized that the landing gear is not deployed.

Next, we constructed two hierarchical levels of the control structure: the first one is the operational process control loop, shown in Figure 9, which shows the interaction of the controllers with the environment, sensors, actuators and controlled process; the context of the system is taken into account at this stage. The second level is the management, as shown in Figure 10, which shows the interaction among the controllers with management entities such as aircraft operators, training etc. The control actions and feedbacks between the entities were identified using the help of the CWA and the requirements identified in the previous phase of this project.

It is important to note that the level of detail at which the communication between humans, the system and environment is specified is much more precise than the communication involving the IA, since the former is well known and exists today whereas the latter is purely based on a high-level of conceptualization of the system communicating with the IA. This way of capturing the details and analysing, we believe, is natural for systems (such as the IA) that are in their conceptualization phase. In fact, by starting at a very high level, the process can begin even before the design and requirements are available and provide actionable results before rework is needed. However, as the IA is iteratively designed and developed, these details concerning IA communication should be updated and the analysis should be incrementally performed.

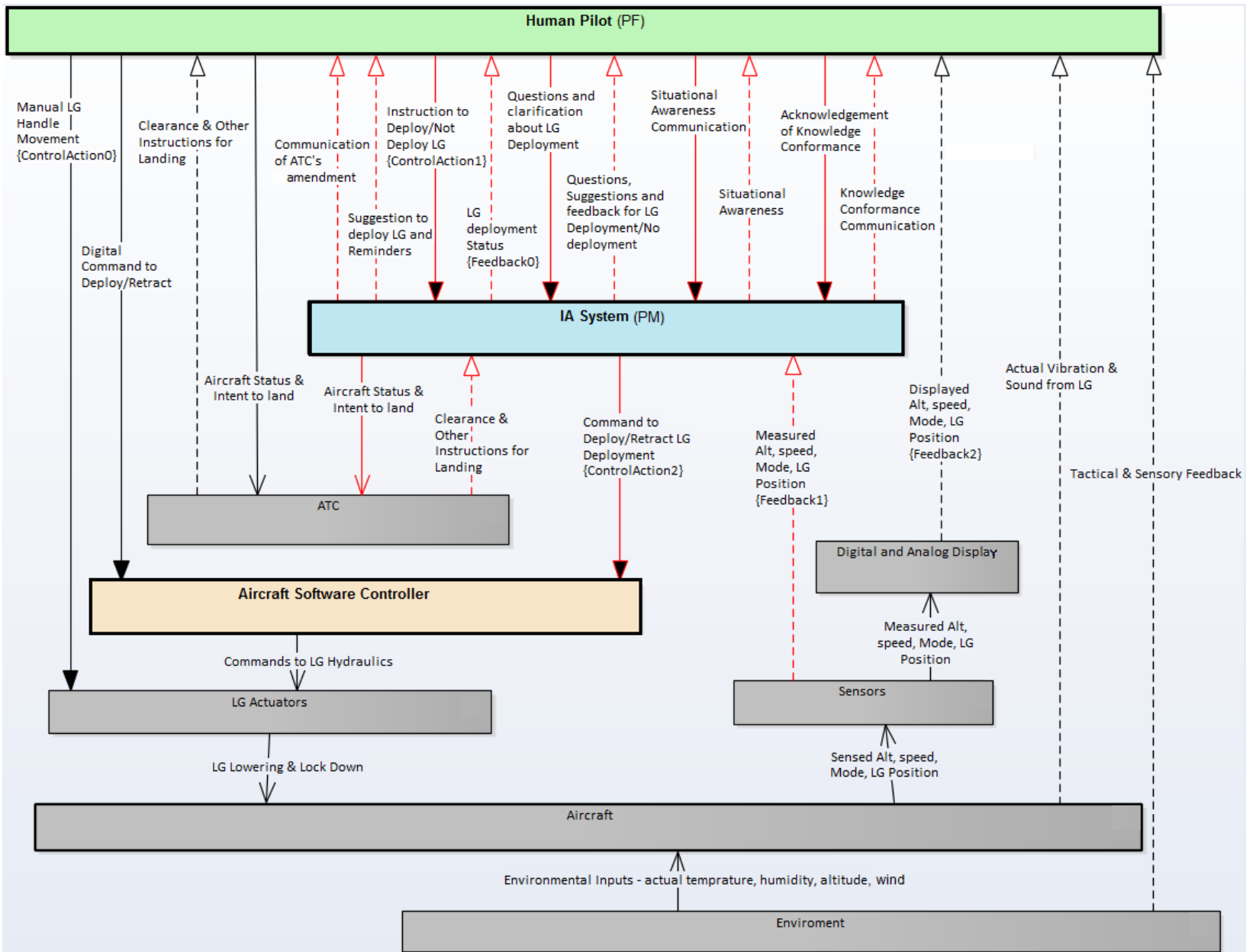


The screenshot displays a hierarchical list of safety events. It is divided into two main sections: 'Losses' and 'Hazards'. Each section is preceded by a folder icon and a triangle. The 'Losses' section contains eight items, each with a red 'X' icon and a description. The 'Hazards' section contains five items, each with a yellow warning triangle icon and a description. A note at the bottom of the screenshot states: 'Note: The hazards and accidents in this figure is presented for illustration purposes only, and is not meant to be exhaustive.'

- Losses
 - «STPA_Loss» Aircraft crashes / catches fire on landing (landing on belly)
 - «STPA_Loss» Aircraft may go around
 - «STPA_Loss» Aircraft lands successfully but is damaged upon landing
 - «STPA_Loss» Aircraft operation is impaired
 - «STPA_Loss» Landing Gear gets damaged
 - «STPA_Loss» Aircraft Speed and Altitude is limited
 - «STPA_Loss» Fuel Consumption is increased
 - «STPA_Loss» Passenger discomfort : Noise and Vibration in Cabin
- Hazards
 - «STPA_Hazard» Landing Gear is not deployed before touchdown
 - «STPA_Hazard» LG deployed during inappropriate aircraft mode
 - «STPA_Hazard» LG Partially Deployed
 - «STPA_Hazard» Correct position of LG is not known to the pilot
 - «STPA_Hazard» All landing gears not deployed at the same time

Note: The hazards and accidents in this figure is presented for illustration purposes only, and is not meant to be exhaustive.

Figure 8: Accident and Hazard List view



Note: The information contained in this control structure is presented for illustration purposes only and is by no means an exhaustive representation of components and communications in an actual pilot-IA collaborative flight environment.

Figure 9: Hierarchical Control Structure - Operational Process Control Level

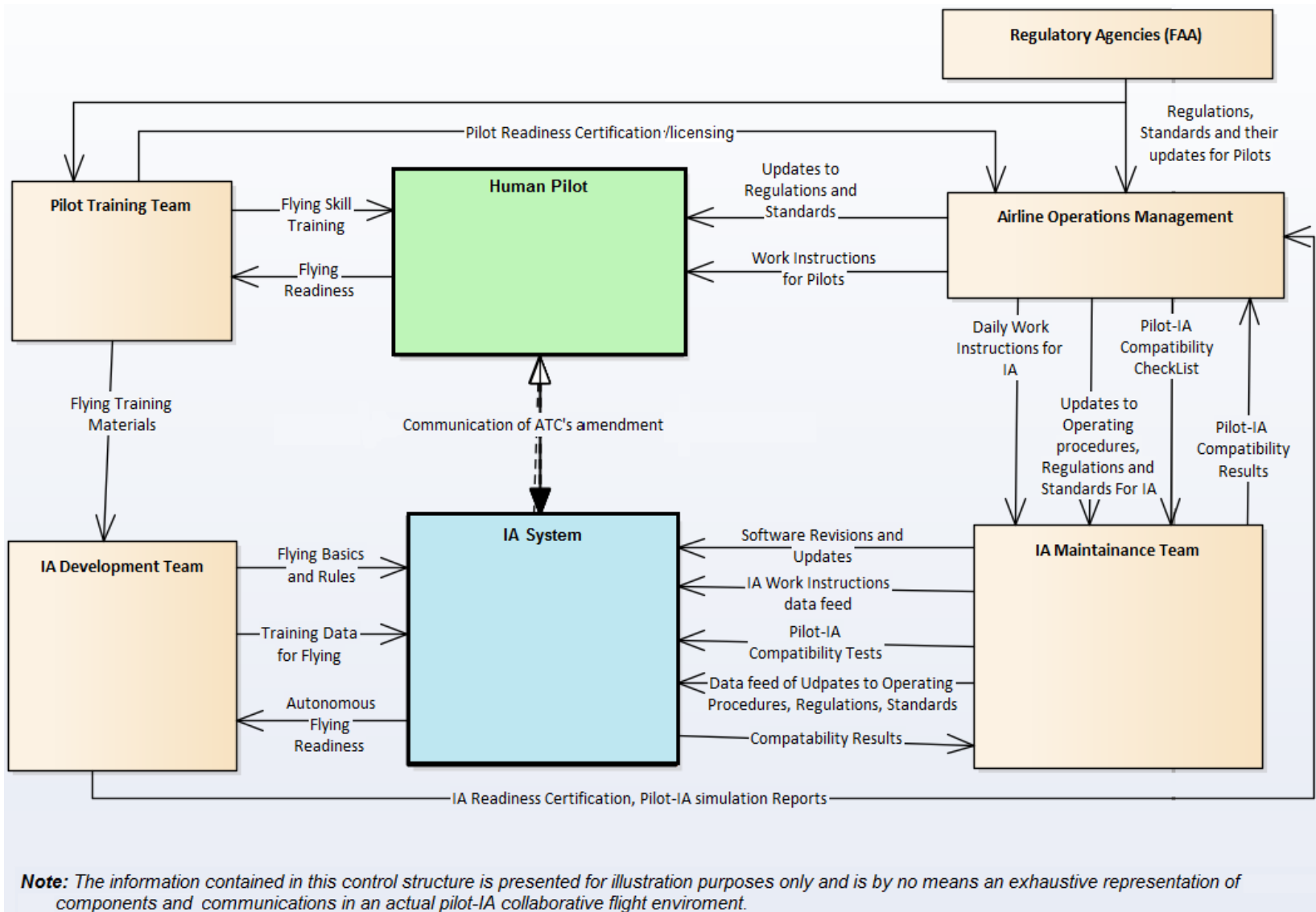


Figure 10: Hierarchical Control Structure - Management Level

Though some control actions and feedback may appear common to human pilot and IA, we intentionally created distinct connections since they would occur through dissimilar means. In Figure 9, the actions and feedbacks that are introduced due to the IA are colored red for visual distinction. For instance, while both human pilot and IA can *command the deployment of landing gears*, the human commands by manual button pushes, whereas the IA can send out messages over the network. At high levels, pilots may acquire flying skills through education and flight training, whereas it could be through some sort of off-line knowledge ingestion or online learning for the IA. Hence, both in flight as well as pre/post flight, the status, regulatory and other procedural updates to humans and IA could be performed through completely different means. All these distinct modes of information exchange introduce newer failure modes such as inconsistency in the information between the pilots and IA. Such an inconsistency could lead to a difference in situational awareness and flight handling strategies, that have the potential to cause accidents.

Once the high-level control structure is defined, an initial set of unsafe control actions can be identified by considering the four ways a control action can be unsafe: A control action required for safety is not provided, provided when it is unsafe to do so, provided too early or too late, provided too long or stopped too soon. Though, STPA provides a tabular notation to document the unsafe control actions (UCA) for Step 1, we recommend a tree-like structuring when identifying unsafe control actions. Such a structure is a powerful means to organize a large amount of related information without losing intellectual clarity. For instance, consider the UCA for the control action “IA command to Deploy/Retract landing gear”, illustrated in Figure 11. The tree-like structuring helped us relate the UCA to the hazard and in-turn the accidents in a much more straightforward way. It also brought to light the repetitions among the hazards and accidents caused by different UCAs. This way of capturing the details, we believe, is useful for analysis such as being able to derive the set of all the UCAs for a specific hazard, and even facilitate automation in the future. While Figure 11 shows only a snippet of the UCA for the control action, the entire UCA is provided as a supporting document. The SAHRA allows us to capture the UCA in the tree structure.

Once Unsafe Control Actions (UCAs) have been identified, the analysis proceeds to identify potential causes of the unsafe control. Similar to Step 1, capturing the causal factors in a tree-like structure helps understand the hierarchy among the causalities in an elegant way. For instance, consider the causal factors for the unsafe control action: “IA does not command landing gear deployment until aircraft touch down”, illustrated in Figure 12. While one of the primary factors for the IA not deploying the landing gear is that the IA is trained to deploy only if a set of conditions (aircraft mode, sensor inputs, etc.) are met, the secondary factor could be that the sensors have failed, and tertiary factors could be that the reliability of the sensors have not been tested properly and the IA is not trained to handle such situations, etc. In typical autonomous flying situations, we believe that it is crucial to be able to exhaustively (at-least as much as possible) hierarchically identify the factors. Again, the SAHRA tool allows such tree representation of the causal factors.

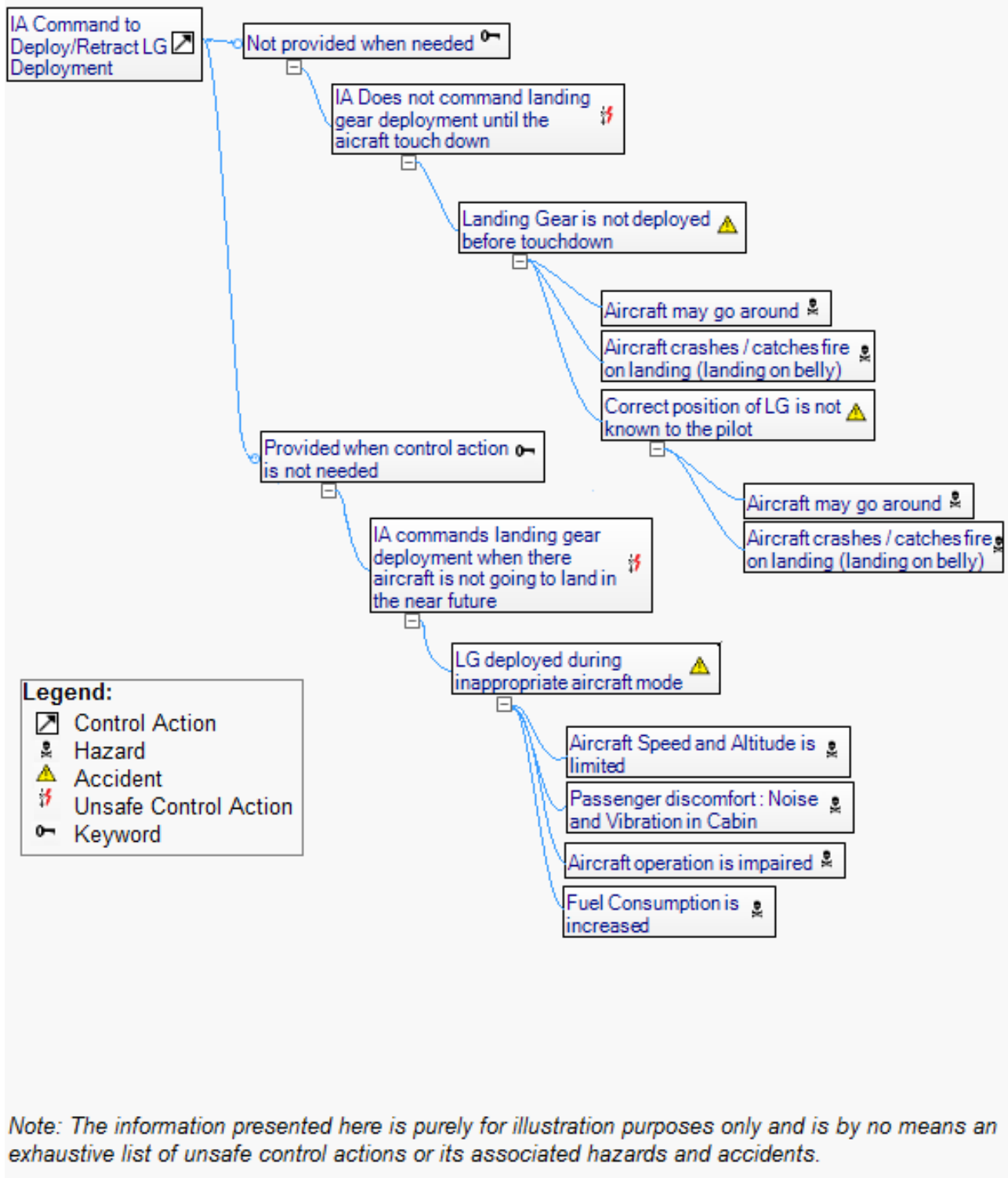


Figure 11: Unsafe Control Actions

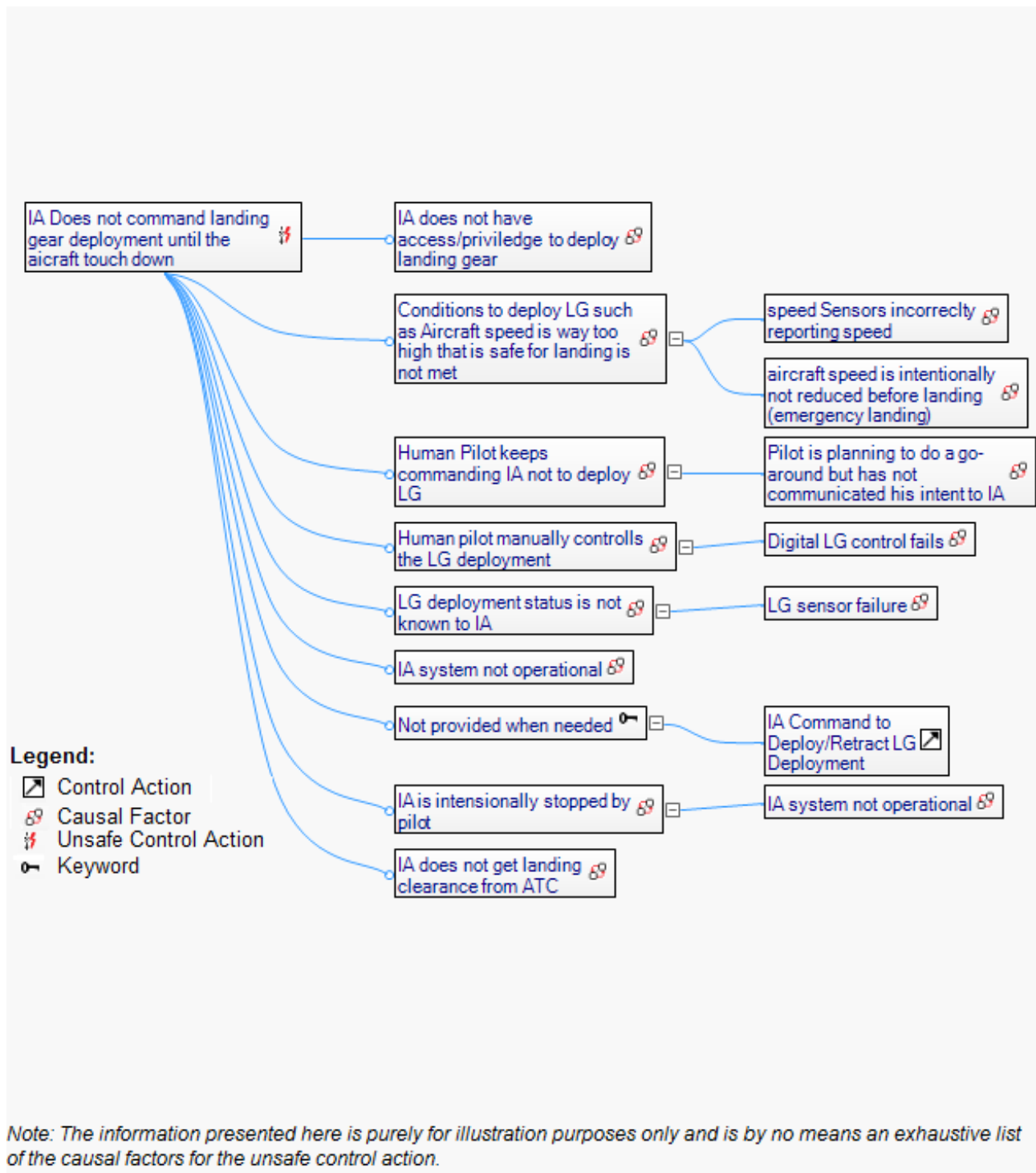


Figure 12: Causal Factors for Unsafe Control Actions

5.4 Discussion

Undoubtedly, STPA is more elaborate than any other existing hazard analysis techniques. With additional guidance and some extensions, we believe that it is the most appropriate method to analyze hazards encountered while introducing an IA system for reduced crew operations.

Including Architectural Concerns: Most of the STPA examples and guidance only illustrate application of STPA at one level (system level), in particular, a single automated controller is shown to control one or more controlled process. But with a more complex architecture like the functional architecture of autonomous systems, the flying sub-system may have several sub-components which interact to control the flight by reading several sensors, computing status and commanding one or more control actions to the actuators. In such cases, determining the hazards by abstracting away the architecture may be inadequate. In fact, the architectural considerations may induce potential hazards or may remove causal chains leading to hazards. Hence, we propose that the STPA should be extended in such a way that its elements have a cohesive relationship with the system's architecture. Such an extension also permits the use of failure models of components, often created during architectural analysis, to identify unforeseen hazards of newly conceptualized systems such as the IA.

Augmenting with System Requirements: We propose to capture the communication among its entities, the role of the functional requirements and the assumptions of those components explicitly in STPA. To effectively perform the causal analysis, it is crucial to know the functional logic that drives the control actions and feedbacks. In fact, the extension by France et al. [54] attempts to capture such logic within human minds. Hence, by capturing such details as much as possible for all the components will reduce the level of subjectivity at which the UCAs and causalities are determined from this process. Further, it may also reduce the number of spurious UCAs and causal factors. Conjointly, by capturing the assumptions that one entity makes about the other, analysts can perform an Assume-Guarantee style of reasoning. Hence, our recommendation is that by extending STPA to include functionalities and assumptions of each entity, using techniques and notations appropriate for the nature of each entity, the precision and rigor of the analysis can be greatly improved particularly for systems without any prior usage.

Considering Multiple Intelligent Controllers: Currently STPA has been only applied to a system that has a human (intelligent) and an automated (not so intelligent) controller. However, when it comes to more than one intelligent and autonomous controllers, one has to account for complexities such as the level of authority and interactions (particularly the problematic ones) among their control actions. The "intelligent" autonomous controller will have its own mental model based on its learning and interaction over time, that may vastly differ from that of a human. For instance, while the human pilot is controlling the landing gear through manual levers (say, he/she suspects a failed sensor), the IA should try to understand or at least question that action instead of simply providing opposing commands. On the other hand, in situations such as the pilot being incapacitated or is found to act with malign intent, the IA should be "smart" enough to override the pilot actions and autonomously

fly the plane. Hence, being able to contextually analyze the causalities of hazards and determine problematic interactions becomes crucial to such systems. However, STPA, in its current form, does not provide guidance to supplement the context (process, time sequence, etc.) in the control loop. Hence, we propose that in order to understand interaction between IA-human teams, we need an extended methodology of STPA. In fact, we need to tie techniques such as simulations to be able to identify such problematic scenarios, since it may not be possible to exhaustively identify them.

Automating the analysis: As mentioned earlier, completely relying on the skill, domain knowledge and subjectivity of the analyst performing STPA is a major hurdle to effectively applying it in practice. The actual control structure for a complex system such as IA-human controlled flights could span many levels of hierarchy and contain several tens/hundreds of components. Being able to manually traverse through the structure and exhaustively identify unsafe control actions and their causalities could be time consuming and error-prone process. Hence, we strongly recommend automating most parts of the analysis—such as deriving causal actions, identifying potential inconsistencies, etc.

Another interesting aspect to consider is the relationship between autonomy and trust in a human-IA cooperative flying environment. While we all acknowledge that pilot Sully’s action to land the flight on the Hudson is the safest action in that situation, how much are we ready to allow the IA go beyond “procedures”—the full physical and belief context of “safe flying”—to categorize it safe or hazardous is still questionable. Being able to understand this trust factor when analyzing the interactions between humans and IA, we believe, also influences hazard analysis.

6 System Architecture Considerations

In this section, we discuss the architectural and system design issues related to IA systems. Before we discuss the specific details and aspects of IA system architecture and system design augmentations, we first review definition of system architecture. Within the US DoD Architectural Framework (DoDAF) [64] architecture is defined as below:

Architecture : the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time.

The Open Group Architectural framework (ToGAF) [128] defines architecture as follows:

- *A formal description of a system, or a detailed plan of the system at component level, to guide its implementation (source: ISO/IEC 42010:2007).*
- *The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time.*

ISO/IEC 42010 [72] referenced above formally defines architecture as:

architecture fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

Reviewing current aerospace guidance (ARP4754A [127]), we find that a formal definition of architecture is not given. But, the development of system architecture is described 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.

With the exception of the implied definition from ARP4754A, none of the above definitions cover functional or behavioral aspects of a system design, which are the most important aspects of an electronic system, particularly for IA. Dori [41], adds behavior to the definition of architecture.

Architecture of a system is the combination of the system's structure and behavior which enables it to perform its function.

Dori's definition better addresses our needs for IA, as it recognizes that it is the combination of structure and behavior that provide the foundation of the emergent architectural properties. Note that Dori's definition of structure is not just physical.

Structure of a system is its form—the assembly of its physical and informatical components along with the long-lasting relations among them.

We will use that definition here. However, Dori's definition of architecture still doesn't make it distinct from "system design". Having two terms for the same concept isn't very useful. A better definition of architecture would emphasize the fact that it is meta design. That is, it is the design for

the design, which includes the rules and constraints for systems designed to that architecture. As an analogy, we can say that an architecture is to design as a constitution is to law. Hence, we define system architecture as follows:

An Architecture comprises the rules and constraints for the design of a system, including its structure and behavior, that would meet all of its established performance requirements and performance restrictions under the anticipated environmental conditions and component failure scenarios.

A system design is an instantiation of an architecture. Thus, system design and architecture instantiation are synonymous. To illustrate the applicability of our definitions, we briefly look at a situation where what previously would have been considered different system architecture designs, are largely equivalent when viewed from our architecture definition's perspective, with the only differences being in implementation details.

In some significant work on fault tolerant systems sponsored by NASA Langley Aircraft Electronic Systems Branch, two systems were developed. These were Draper's FTMP [70] and SRI's SIFT [138]. While these two system were often portrayed as being the yin and yang of fault-tolerance system design (one instantiating its key architecture features in hardware and the other using software) [102], one can create an architecture statement that equally describes each of these systems:

The system consists of a set of processors that can communicate among themselves. The communication design includes the exchanges and other mechanisms needed to make the communication tolerant to one Byzantine fault. Any processor can communicate with any other processor as a peer. From this set of processors, a triad of processors are elected to form an oligarchy which configures and controls the rest of the system. This is a form of hybrid triple modular redundancy.

Thus, hardware and software are media for expressing an architecture as a system design implementation and are not an architecture distinction in themselves, unless there specific system requirements for such a distinction.

As illustrated in Figure 13, an architecture's rules and constraints can be applied to *multiple levels* of design refinement along three orthogonal aspects: behavioral, logical, and physical. Thus, an architecture applies to more than just the top level of a system's design. The behavioral aspect often is called "functional". Many so called architecture diagrams are really a system diagram that is a confused mixture of each of these three aspects. For example, a diagram may show gyroscopes (physical aspect) connected to something called "Inertial Measurement Processor" (logical aspect) that is then fed to something called "Navigation" (behavioral aspect). Some key pieces of an architecture include the philosophy of how fault tolerance is done. For example, is fault detection done via bit-for-bit exact-match comparison, threshold comparison, inline coding (e.g., CRC), etc. Another key piece is the timing of redundant components. Is that lock-step synchronous, fully asynchronous, plesiochronous, etc. These pieces can apply to multiple levels of design refinement.

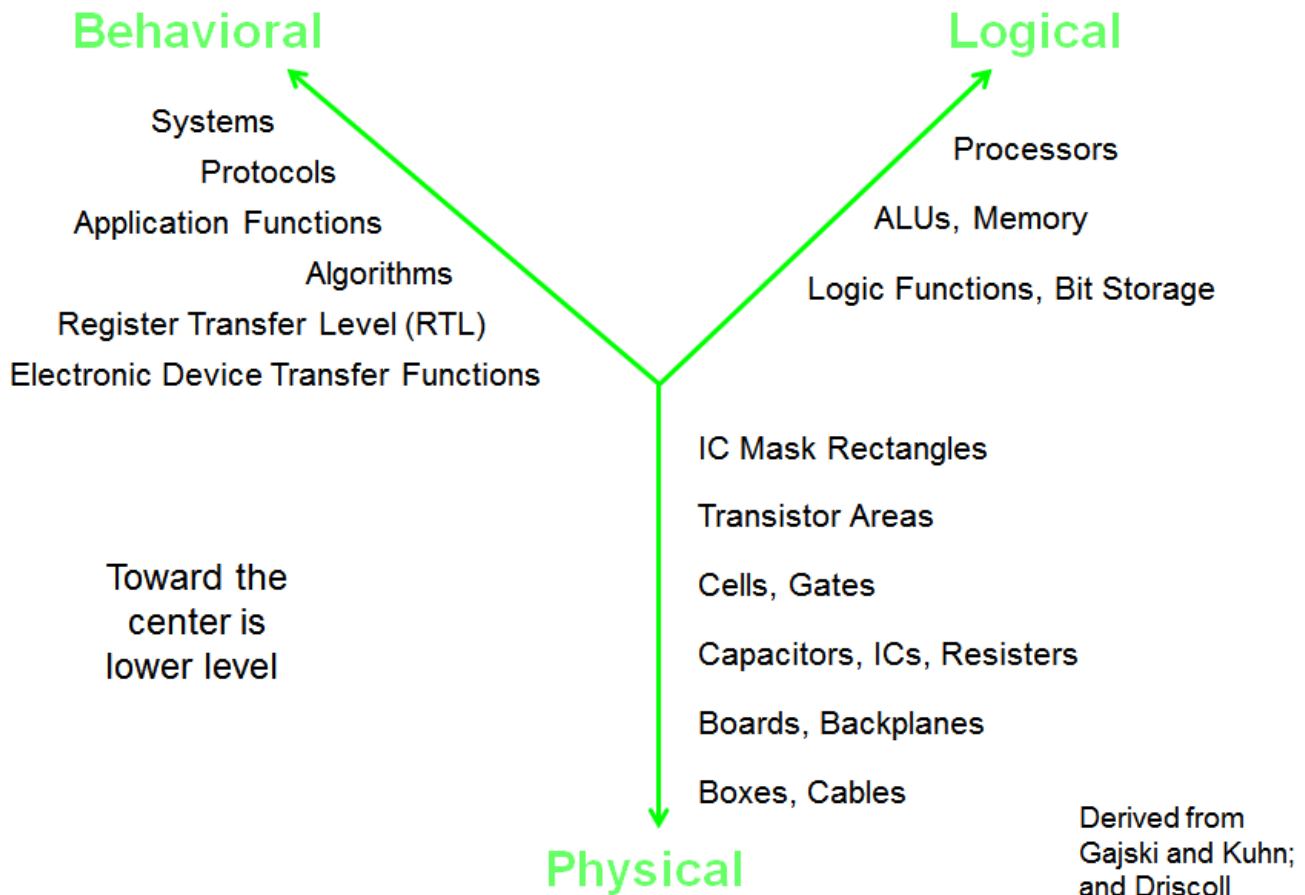


Figure 13: Orthogonal Aspects of System Design (Architecture Instantiation)

6.1 Establishing the System Context

Within the broader system context of our definition for architecture, the issues relating to IA system augmentation may be better addressed than narrower views of architecture or fixations on system components. First, system hazards and safety contributions may not be evident at the individual component level. Therefore, as argued by Leveson [85], system safety assessment approaches that focus largely on component reliability and failure mechanisms may be challenged by the complexity of more software intensive systems. Indeed, one of the biggest challenges to achieving autonomous systems with assured safety is the fact that safety is an *emergent* property in such complex systems. Therefore, targeting safety and design assurance at the component level may, in general, yield sub-optimal results. On the contrary, the complex behaviors of human-system interactions and multiple component and environment interactions of non-faulty components may dominate the safety arguments. With respect to IA system components that have been trained using machine learning technologies, this concept of safety as an *emergent property* become even more prominent. This is because the component behaviors themselves can be emergent from the training data sets. Leveson

argues that a more holistic *systems thinking* perspective is required to mitigate the risk relating to the increasing complexity of software intensive systems. She proposes the STPA (System-Theoretic Process Analysis) as a safety analysis framework to support a more holistic systems thinking perspective. The STPA approach provides a systematic process to identify the constraints on component behaviors that are required to mitigate system and mission level hazards. The hierarchical nature of the STPA technique enables this approach to be applied at many different levels of the system architectural and behavioral concerns as illustrated in Figure 14.

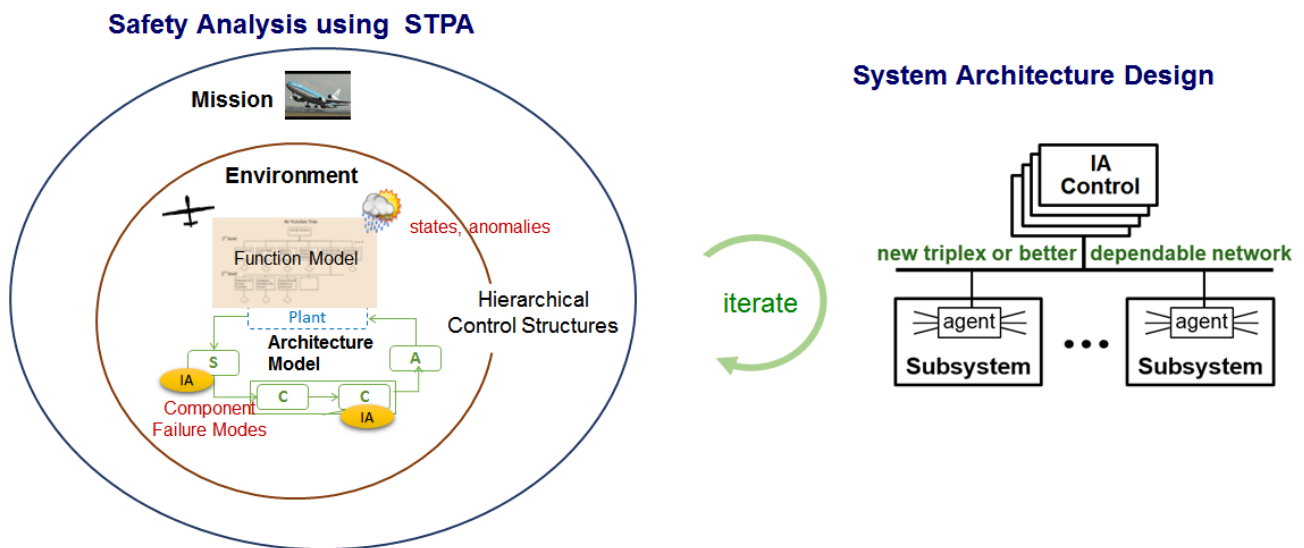


Figure 14: Architecture Refinement using STPA

This provides a capability to examine the influence of the IA components from several integrated mission and architecturally aware perspectives. Figure 14 illustrates the conceptual idea of architecture refinement. As shown in this figure, the system architecture hosts the *functions* that the system provides, including interfaces to the environment (e.g., sensors and actuators). This is captured in the Function Model, which is a parametric hierarchical decomposition of functions. When the functions are traversed using the hierarchical control structures within the STPA method, the logical and physical components can easily be mapped to the architectural elements.

A key aspect of this approach is that it supports the analysis of system-level impacts of a component’s causality. This can be established and addressed within the context of its use. For example, the behavior of a classifier may be evaluated against its impact w.r.t. the sensory and control flow within the STPA models and hierarchical control structures. Traversing up through the hierarchy, this flow is evaluated for impact on the system hazards and mission objectives. We believe that this “system-thinking based” integrated approach is essential in grounding and optimizing the system assurance argument. The wider contextual perspective derived from the STPA modeling provides a foundation to mitigate many of the known challenges related to machine learning, such as those enumerated by

Amodei et al. [10]. The integrated perspective afforded by the top-down system thinking approach will provide a suitable environment to identify and introduce constraints to mitigate “reward hacking” and related “negative side effects” of learning algorithms.

6.2 Guided and Principled Architectural Refinement

In her book, Leveson provides some good insights into safety-derived design. However, specific strategies for the architectural refinement of STPA-based results are not fully defined. In adjacent research [141], funded by the FAA, we have proposed an extension to STPA that better formalizes the additional steps which may assist with such architectural refinement.

The driving need for such *principled architectural refinement* was made apparent following the review and analysis of several aircraft incidents that were caused due to system errors that should have been mitigated or prevented by architecture design. These incidents were largely due to missing derived architectural requirements, such as the input congruency and state agreement exchanges required for Byzantine fault mitigation. Our prior research also uncovered other errors and system risks related to inconsistent architectural redundancy reasoning with respect to system availability and system integrity requirements, for example, the inconsistent assumptions in the application of system redundancy and inline fault detection mechanisms. Many, if not all, of the aforementioned challenges also exist in IA systems. In such systems, architectural issues such as Byzantine resilience, coverage/scrubbing for latent fault detection, distributed redundancy management, and signal validation, all need to be addressed using “systematic architectural thinking” in a manner similar to the “system control theoretic thinking” that is deployed within STPA. Hence, we believe these suggestions and guidelines established for principled architectural refinement may also be very relevant to the needs of IA augmented architectures. In addition, we believe that the wider environmental and operational contexts, that can be established within the STPA-based analysis, may also help establish the criteria for pervasive monitoring (details are in section [subsection 6.3](#)). The idea behind pervasive monitoring is to constantly monitor the state of the aircraft and the inferred state of its environment against models of “safe flight” that are model-based representations of the regulations, information, and advice recorded in sources such as the “Aviation Handbooks & Manuals” maintained by the FAA [46]. We say ‘models’ rather than ‘model’ because we anticipate the full model will be the composition of several component models: some will be specific to the type of aircraft, while others are community-supported and generic to all flying activities.

Establishing the system environmental and causal influence of control actions within the wider operational contexts, utilizing STPA-based techniques, may additionally benefit the derivation of the derived safety properties. From such a system context, suitable component behaviors and models may be derived. Validating these models, individually and in composition, will be a significant—though, we believe, feasible—undertaking. In comparison, monitoring the execution of an individual flight against the composed model seems like straightforward engineering. However, it does raise some architectural issues and pervasive-monitoring-based mitigation approaches, addressed next in Section [6.3](#).

6.3 System Architecture and Pervasive Monitoring

There is a field of “runtime verification” with a vast literature [33], and a sub-field of “distributed runtime verification” with a large literature of its own. Many of these papers are concerned with monitoring rather complex formulas in a temporal logic; those concerned with distributed systems deal with complexities where not all variables can be monitored at one site. For distributed systems, there are issues of coordination among multiple local monitors and the possibility of inconsistency if one monitor sees an event and another does not.

In a large (AT) IA aircraft, it is reasonable to assume that the underlying system has highly-reliable data communication and coordination (otherwise there are far more urgent matters to deal with than those addressed here). So, all the basic data required for monitoring may be assumed to be available at some (possibly replicated) monitoring site(s).

BRGA aircraft, on the other hand, may not have the same guaranteed and complete data communications as AT aircraft and some of the issues of conventional distributed runtime verification may arise. However, we think these are best addressed as part of the general revisions to BRGA architectures required for IA systems, as discussed elsewhere in the architecture report associated with this effort.

Here, we will assume that all required data is available at a single site and that monitoring this data against models of “safe flight” is straightforward and does not involve the difficult liveness cases of temporal logic. There are, however, issues concerning exactly what it is that is monitored.

Monitors for traditional avionic functions focus on behavior of the function. For example, the monitor for, say, a fuel management system tracks the inputs and outputs (and possibly internal state) of the fuel management system and checks that it satisfies critical properties derived from its requirements. This arrangement, as applied to IA control, is portrayed in Figure 15.

As we discussed in our requirements and assurance reports, it is possible that IA functions will be developed in a conventional way, and could therefore be monitored like a conventional function. But it seems more likely that at least some IA functions will be derived by machine learning or by online synthesis and these may lack conventional requirements. Furthermore, one of the purposes of IA system capability is to mitigate human pilot error, so it is not only IA behavior that must be monitored, but that of the human pilot as well. From these considerations, we see that the purpose of pervasive monitoring is not to second-guess IA behavior but to ensure safety of the aircraft as a whole. Thus, pervasive monitoring does not monitor the IA function, but the entire state of the aircraft and its environment. Its purpose is to ensure that the aircraft is safe, both with regard to its own systems and aerodynamic state, and in relation to the world around it (i.e., the “rules of the air”). This arrangement is portrayed in Figure 16.

The architectural consequences of this analysis are that pervasive monitoring must be independent of all other aircraft and human functions; specifically, its failures must be independent. This means that its design and implementation should be diverse from other aspects of aircraft automation (this is

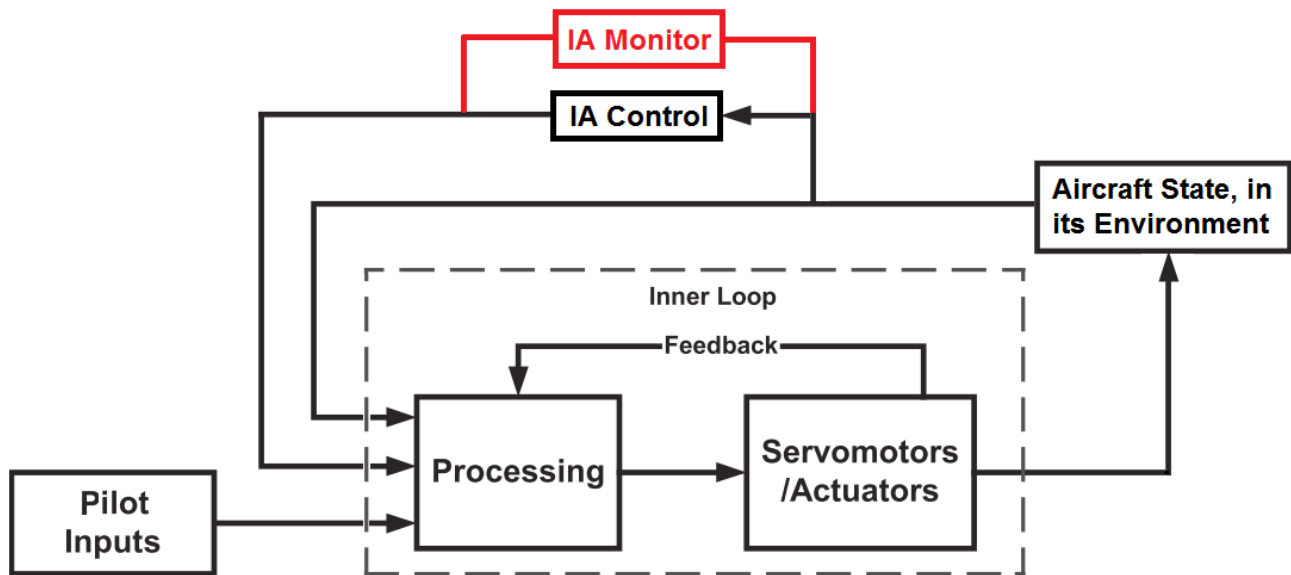


Figure 15: Conventional IA Monitor Architecture

largely ensured by its model-driven character) and physically separated from them.

Probabilistic Analysis

The considerations above can be expressed in a probabilistic model for the risk (i.e., product of cost and frequency of failure) of a pervasively monitored aircraft. There are two kinds of failure that must be considered:

Type 1: this is a failure of the aircraft (including human pilot and IA automation) leading to a potentially unsafe condition, that is not detected by the monitor.

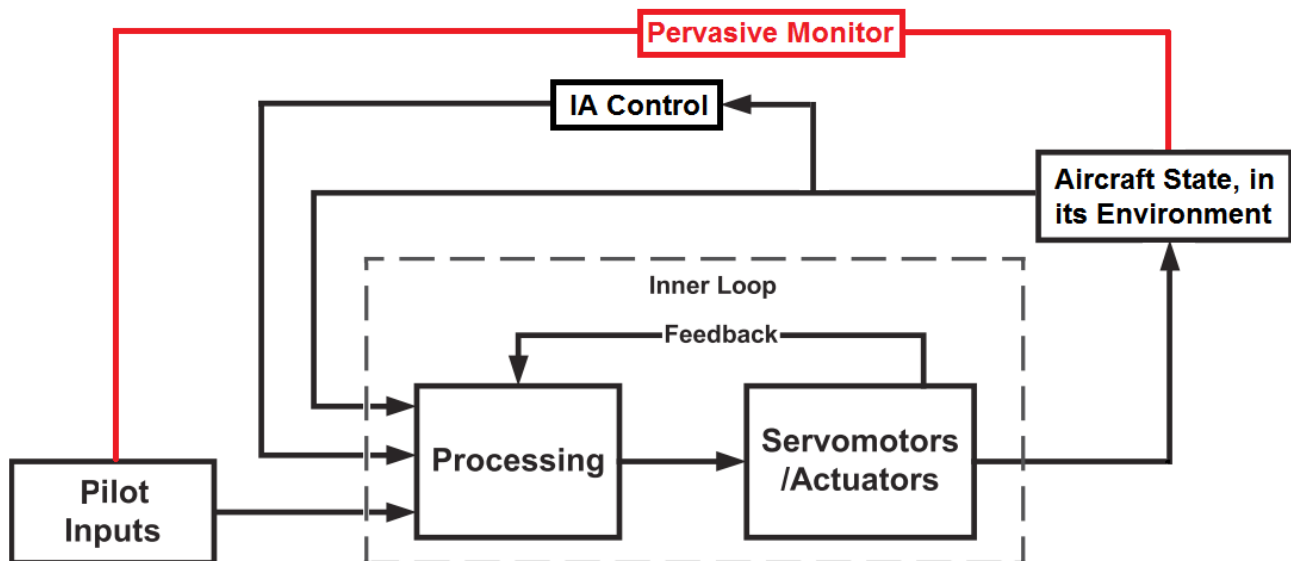


Figure 16: Pervasive Monitor Architecture

Type 2: this is a failure of the monitor alone, that causes it erroneously to signal a potentially unsafe condition.

The total risk is the sum of the “cost” of each of these, times its frequency or probability as shown below.

$$\begin{aligned} \text{Pervasively monitored architecture risk per unit time} & \quad (1) \\ & \leq c_1 \times (F_A \times P_{M1}) + c_2 \times P_{M2}, \end{aligned}$$

where c_1 is the cost of a Type 1 failure and c_2 the cost of a Type 2 failure, F_A is the probability of failure of the aircraft (including human pilot and IA automation), P_{M1} is the probability of a Type 1 failure by the monitor and P_{M2} its probability of a Type 2 failure.

Interpreting this formula raises some interesting questions. We assume the cost c_1 is extremely high (a potentially catastrophic failure condition), which requires the other factor in the first product to be extremely small (on the order 10^{-9} per hour). Current aircraft achieve this, but we assume some of that is due to mitigation by two pilots and some is due to the conservative design of current automation. An IA system and single pilot operation challenges both of these contributions. On the other hand, many if not most, current incidents and accidents are attributed to pilot error, which could be mitigated by IA automation. We might suppose that an IA aircraft in single pilot operation will do better in some cases and worse in others than a conventional aircraft with two pilots. The requirement on the monitor, then is to make up any shortfall in the overall safety of the IA aircraft in single pilot operation. Plucking a number from the air, this suggests a requirement of the order $P_{M1} \leq 0.1$. Simplifying a little from [86], this could be achieved if we are at least 90% confident in the *perfection* of the monitor. This seems feasible, but one way to achieve it is to make the monitor conservative, which will likely raise the likelihood of Type 2 failures. If we assess the cost c_2 as low (e.g., a warning that the pilot can override), then there is a danger that real warnings will go unheeded. Hence, we should assess this cost as fairly high, and that imposes a requirement that P_{M2} must be small. If we allow one false alarm per aircraft per year, this requires roughly $P_{M2} \leq 10^{-4}$.

Summary for the Pervasive Monitoring Idea

Logical and architectural considerations indicate that a pervasive monitor should be an entirely separate aircraft function that monitors the state of the entire aircraft, including IA automation and actions by the human pilot. Its need for reliable access to all aircraft data should be satisfied by AT aircraft, but should be added to the architectural requirements for BRGA aircraft.

Simple probabilistic analysis suggest we need about 90% confidence in perfection of the monitor but that its probability of Type 2 failure must be as low as $P_{M2} \leq 10^{-4}$, which is a demanding combination.

6.4 Overview of the Approach for System Architecture when IA Systems are Used

Augmenting current systems with IA capabilities has a fundamental, multidimensional impact on the system architecture from the safety perspective. Approaching this from a perspective of “extending the architecture with an advanced subsystem” necessarily will be inadequate. Rather, the impact considerations must re-evaluate the current notions and assumptions about safety and dependability—taking into account the entire context of the environment, safety arbitration and degrees of authority/autonomy, and human-IA interactions. In particular, the following issues are relevant:

- The use of autonomous capabilities in reduced crew operations bring new issues in terms of environment sensing, state awareness, decisions making, control authority sharing, and safety arbitration between humans and IA functions. The architecture needs to provide mechanisms to enable this.
- The current approach to dependability (availability, integrity, security, etc.) is based upon component reliability and “fault detection isolation and recovery” mechanisms at the architecture level. A new context must be added to this approach to account for control authority sharing and safety arbitration by the IA functions.
- A top-down safety analysis must be performed to comprehensively assess and refine the architecture design for mitigating safety hazards and provide the dependability in the new context.

6.4.1 System Dependability Requirements in the Broad Context

It is meaningless to refer to a system as “autonomous” or “semi-autonomous” without specifying the task or function being automated. Different decisions have different levels of complexity and risk. Any given machine might have humans in complete control of some tasks and might autonomously perform others. For example, an autonomous operation may only be incorporated for normal flying conditions, omitting operations where the pilot might have to take control during abnormal conditions. Hence, the system may only be autonomous with respect to some functions. From this perspective, it is more fruitful to think about “autonomous functions” of systems, rather than characterizing an entire vehicle or system as “autonomous”. This clarifies the confusions about “dependability” discussions; one can now set varying levels of dependability criteria depending upon the functions and context.

Also, the extent to which a function is made autonomous has a huge impact on the architecture of the system. The level of autonomy refers to the degree transfer of control from one part of the system (e.g., ground control with humans) to another. An increase in system autonomy is typically achieved through an increased integration of automated functions in combination with a transfer of execution authority from the human operator to the system. A common misconception is that an increase in intelligence of a system increases the level of autonomy. In most cases, the intelligence or complexity of the machine is a separate concept from the tasks it performs. Increased intelligence

or more sophisticated machine reasoning to perform a task such as advanced object recognition does not necessarily equate to transferring control over more tasks from the human to the machine (e.g., object avoidance). Similarly, the human–machine command-and-control relationship is a different issue from complexity of tasks performed.

6.5 Impact of System Concepts and Design Choices on Architecture

6.5.1 Human-IA System Interactions and Control Authority Considerations

Autonomy and authority issues will be exacerbated in any environment where more sophisticated automation will be employed to improve the overall safety and efficiency of operations. Just as we are developing a verification and validation paradigm for evaluating the effect the IA system has on human performance, we must also derive a standard approach for creating scenarios that will evaluate a broad and thorough set of potential autonomy and authority issues. The placement of controls and displays, as in a traditional flight deck, is critical to optimizing the human pilot’s workload and task management. Autonomy is considered to be inappropriately allocated if the agent (human or IA) lacks the necessary information or controls to perform the task. Controls and displays should be arranged in such a way that considers the levels of authority and autonomy between the human and the IA for each task type. Displays should dynamically update to indicate the status of the task or activity being accomplished by an IA system and should highlight the task order and prioritization for the pilot.

Task management is a key component of prioritizing activities efficiently and effectively between the IA system and the human pilot. The dynamic nature of the operational environment can lead to inappropriate task and function allocation, as time or workload constraints may make it difficult or impossible for a human to perform certain tasks. Function allocations that result in unevenly distributed workload, communication, and coordination demands deserve particular consideration. Poorly defined or ambiguous autonomy may also lead to poor human-IA interaction. “Ambiguous autonomy” results from failure to account for the inherent capabilities and limitations of an IA system. There are a multitude of recorded incidents in which the pilots failed to account for differences among various autopilot modes (e.g., the loss of performance envelope protection in certain modes). Likewise, an IA system may fail to consider the variable nature of a human’s capabilities and limitations. For example, during periods of low workload a pilot’s vigilance may wane and their ability to notice subtle changes to the environment deteriorate. On the other hand, during periods of high workload, the pilot’s attentional channels narrow and their ability to process complex information deteriorates. In each of these cases, at what point should an IA system attempt to take on additional tasks or suggest changes in roles?

Just as with ambiguous autonomy, ambiguous authority, or crew member roles and responsibilities, can be a serious problem for human-IA systems. While autonomy can be defined as the *capability* to perform functions/actions independently, authority refers to the *requirement* for executing a process associated with those functions and actions.

Many issues between humans and automation result when there is ambiguity over who has explicit authority or when authority is transitioned from one agent to the other. This ambiguity often leads to “authority surprises” in which a crew member unexpectedly performs an action that crewmates did not anticipate (e.g., an IA system makes a sudden trajectory change that the pilot did not expect).

Ambiguity regarding who has authority can also lead to “authority gaps” wherein someone has the requirement to take action, but doesn’t act. Authority gaps can be a result of authority not being explicitly assigned or it can be due to mode confusion. Mode confusion can occur if the human pilot or IA system is unaware that the other has changed modes or has transitioned the controlling authority back to the other.

Ambiguity can also result from “authority overlaps” if there is confusion over which agent has authority. For example, performance envelope protection is intended to help pilots and automation to not exceed structural and aerodynamic operating limits. However, there have been incidents in which this protection has overridden the authority of the flight crew and either induced the incident or prevented the pilots from recovering from it.

Control authority and autonomy paradigms can be preserved using safeguards such as lockouts, where deemed necessary. An example of an effective lockout is when multiple actions are required for the human to override an IA system when the IA system is the currently designated authority (e.g., PF role). Other ways to mitigate autonomy and authority ambiguity is through thorough and adequate training for the human pilot, clear and concise SOPs, salient display elements indicating who has control authority (particularly for safety-critical controls), and effective communications between the crew as necessary to clarify changing roles and responsibilities.

Listed below are more key considerations for the assignment and dynamic assessment of control authority:

- How do we decide who is in charge? How does this decision-making change based on the scenario and operational environment?
- When is it safe for authority to be transitioned?
- Is there a way to assess the shared situation awareness of the crew when there is a request for control authority?

Example: The human pilot was outside the flight deck when there is a request from ATC to change altitude on approach. Upon returning to the flight deck, the pilot observes the aircraft climbing and requests control to return to the approach altitude. The IA system will need to:

- Be aware the pilot was not present/not listening to ATC communications
 - Withhold authority transition until the pilot has been briefed
 - Establish an “agreement” with the pilot as to the proper next steps
- Can the pilot can override the IA system?
 - In which situations (if any) may the pilot be locked out?

- How will the IA system detect a “failed” or “malicious” pilot? Example: The human pilot is acting as PF, and suddenly experiences a seizure that causes him to kick the rudder hard-over. The IA system will need to:
 - Promptly detect and investigate the anomalous control condition
 - Identify the incapacitation (within an acceptable level of certainty)
 - Assume authority over the primary flight controls
 - Alert the pilot that it is taking control (and continue to provide alert until acknowledged)
 - Potentially alert the ground and other aircraft (MAYDAY, PAN-PAN)
 - Monitor for further pilot input or requests to assume control. Establish pilot competence to re-assume control
- What safeguards will we have to reduce the risk of the IA system misclassifying a pilot as “failed” or “malicious”?

An important observation is that the answer to questions about whether an IA system, the onboard crew, or some authority outside of the aircraft has the ultimate decision authority for any given situation must be defined before that situation unfolds. Otherwise, who has the authority to decide what the situation is and what the rules are? Whoever/whatever has that decision authority is the ultimate authority. There can be only one ultimate authority for any situation. This means that any IA system designed to protect the aircraft from an malicious crew (or crew incapacitation that is indistinguishable from being malicious) is mutually exclusive to a design that allows the crew to override some IA system failures. There are no exceptions to this mutual exclusion. One cannot design an system that can handle both of these scenarios; it has to be one or the other.

6.5.2 Situational Awareness: Additional Sensors and Processing Considerations

There are several questions regarding the information needs for an IA system:

- How much and what data does an IA system need?
- How much different is that data from a human pilot?
- For an IA system to be “more reliable” and trustworthy in decision making, does it require “more” access to sensor data?
- How often must data be updated?
- Is it a push or pull system? (this applies to both control actuation options)

Pilots do “multiple sensor fusion”, e.g., detecting smoke, noise, vibration, etc. This includes non-traditional sensing (microphones, gas detection, vision video). This is a high bandwidth and complex operation.

How does an IA system get all the data it needs, when it needs it, and the quality it needs? How does the IA system provide actuation and information outputs? The architectural issues relating to these questions are covered in the next sections.

6.5.3 IA System Integration into Avionics

The methods that IA systems can use to integrate into an aircraft's systems can be placed into several categories. These categories are, in rough order of increasing invasiveness and complexity are:

- Exogenous: an interface in the form of a robot that is not permanently affixed to the aircraft
- Network: intercepting signals available on the aircraft's data networks
- Remote Agent: insertion of an IA system's agents in all IA system influenced boxes and an interface added to a new IA data network
- Porcupine: point-to-point connections from an IA oligarchy to all points where IA needs access

For IA integration methods other than exogenous, do we need to add special actuators to provide an electronic interface for an existing mechanical interface, e.g., the "gear down" lever. Or, do we replace these interfaces with a purely digital interface (electrical or optical)? What is the required redundancy for such interfaces?

Exogenous Interface

Among the IA interface methods, exogenous interfacing is the one that somewhat matches the commonly held view of a humanoid robot. Its input/output modalities are limited to just those of a human pilot. This is largely visual and mechanical, but could include audio as well. The only attachment that a purely exogenous IA system would have to an aircraft would be some mechanical attachment to hold it in place. Preferably, this attachment would be only the existing method to hold a human pilot in place, i.e., the seat and seat-belt system.

An example of an exogenous IA system design is the one created for the Defense Advanced Research Projects Agency (DARPA)'s Aircrew Labor In-Cockpit Automation System (ALIAS) program by Aurora Flight Sciences. A video of this device can be seen on YouTube (<https://youtu.be/om18cOWFL3Q>). The current ALIAS prototype is too slow to be useful. One can expect that performance would be improved with more powerful actuators for the arm. However, when it is made fast enough to be useful, the arm will have enough energy to become dangerous. It could smash the windows, the pilot, and/or controls. It could pin the pilot, preventing the pilot's access to the controls. Even without the hazards of more powerful actuators, this arm can be dangerous. Clearly, a design error or system failure could cause the arm to do an incorrect action. More subtly, the arm could do inadvertent hazardous actions as a byproduct of correct intended functions. Such a hazardous byproduct action can be seen in this video. For 4 seconds, starting at time 1:14, a portion of the arm pushes the yoke forward and also rotates it clockwise, as shown in [Figure 17](#). Such inadvertent movements of the yoke can cause the autopilot to disengage. This scenario would fully disconnect the autopilot, even for those autopilots where the pitch and roll axes are disconnected separately. Inadvertent bumping of the yoke and the subsequent autopilot disengagement has led to the loss of at least two commercial airliners.



Figure 17: ALIAS Inadvertent Hazardous Action

The use of exogenous IA system raises several questions:

- Will there need to be different configurations of switches, interfaces, etc. and/or additional actuation mechanisms?
- Can all controls be handled by a mechanical robot IA system?
- How do you “safe” a robotic arm?
 - Force fight?
 - Electronic monitoring and disabling?
 - Make the reach of an arm short enough not to block a pilot’s view, knock over coffee, hit the pilot, or anything else it shouldn’t?

Network Integration

This IA system integration method attaches to aircraft systems only through existing data network links. Its invasiveness and complexity is limited because no existing avionics box has to be “opened up” to make internal changes.

One attachment method would be to disconnect each existing fiber-optic or wire cable, which contain signals that the IA system needs for input or output, and insert a tee at the point where the existing cable is disconnected. This tee would consist of two connectors of opposite gender, which would mate with the existing cable connectors and would provide signal flow through for those signals not intercepted by the IA system, and a third connector or cable pigtail that would attach to the IA system. This tee also could include some electronics to act as a repeater for the cable’s signals. It is highly likely that such repeater electronics would be needed for any fiber-optic cable. This type of attachment is sometimes called “lump in the cable”; or, more often, the misnomer “bump in the wire” is used.

Another attachment method is applicable only to data networks that have a central point through which communications flow, e.g., a switch, a hub, a passive star, or a bus medium. For this type of attachment, the communication generally would be overridden rather than intercepted. The difference between overriding and interception is whether the flow of the original communication is blocked or not. For an interception, the IA would block the original communication and replace the blocked communication with its own version. For overriding, the original communication is allowed to flow normally; but, the IA would follow up with overriding communication that would “countermand” the original. The more unusual cases where this centralized method of attachment would use interception rather than overwriting would include purposely transmitting at the same time as the original source, causing collision errors that block reception of the original transmission.

Given that most avionics suites have a non-homogeneous data network architecture, it is likely that the network method for inserting an IA system would include both types of attachment described here.

While the invasiveness of this method is limited, it is not zero. Its primary invasiveness is to increase the latency of signals within existing data network links. Thus, one design goal for this IA system interface is to minimize this latency. The total latency for all existing time-critical signals intercepted by the IA system will have to be reevaluated to make sure that system requirements continue to be met.

There also could be a hybrid IA system actuation system that includes some exogenous actuators along with the network intercepts.

Centralized Porcupine Topology

In the “Porcupine” integration method, individual signals go out from the IA system override boxes that provide the override capability to all the points in all the other subsystems where the RCO must intercept some existing signal. The name “Porcupine” comes from the fact that there might not be enough surface area on the IA system boxes to accommodate all the possible signal lines.

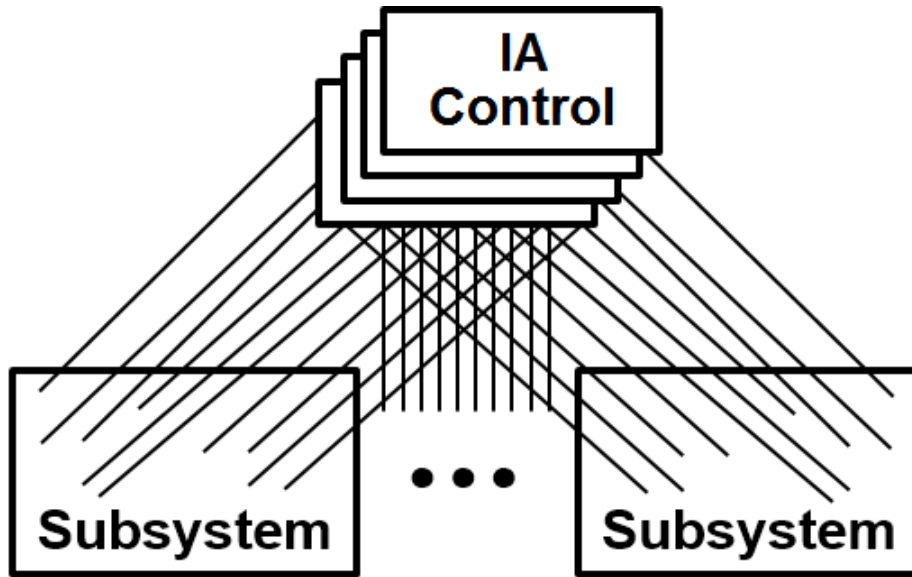


Figure 18: Centralized “Porcupine” Topology

Remote Agents Topology

In the Remote Agents integration method, the mass of Porcupine wires are replaced by a new high-dependability network that connects the IA system override boxes to remote agents within each of the other aircraft subsystems. The only difference between this Remote Agents integration method and the Porcupine method is the structure of the signal interconnects.

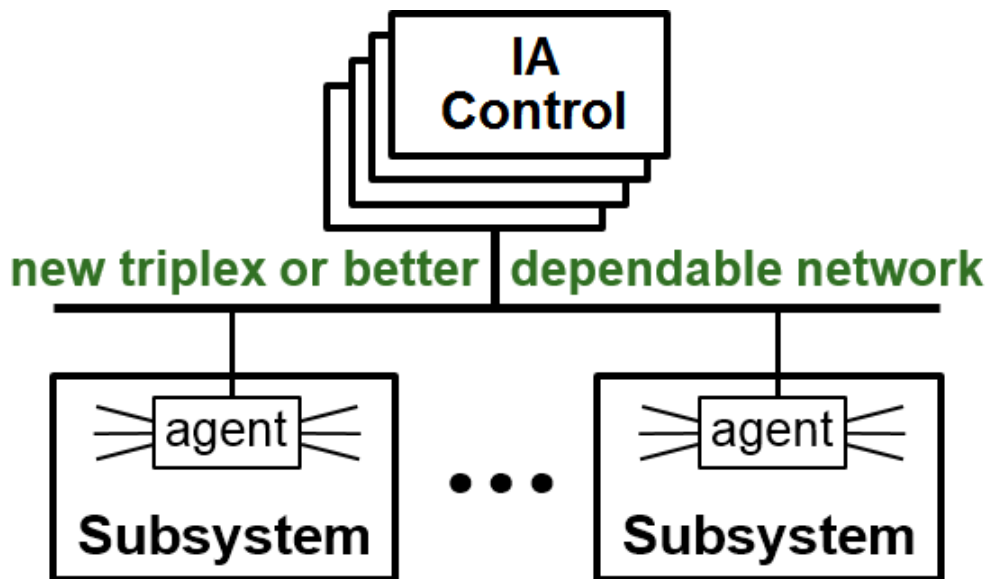


Figure 19: Remote Agents Topology

6.5.4 Distributed Vs Centralized processing

The conventional avionics system design approach is to distribute processing of sensed data around the system. For example, a front camera captures pixel data and turns that into classified object data, such as a bird. Or an ultra-wideband radar sensor that turns radar reflections into distance and angle data about another flight in front. In these cases, a high level “object” data is passed to the system’s autonomous component and processed further to make decisions about altitude, angle, etc. This distributed processing model offloads raw data analytic to components, so that the sensor fusion job becomes simpler to a certain extent; the autonomous component only needs to deal with high level object data. This means balanced power management, easy fault localization, and isolation for safe autonomous systems.

Another topology under consideration is the centralized processing model. In this case, the camera or radar sensors capture raw data and passes that raw data to a centrally located high-performance machine to conduct object level processing. This means the raw data analytics lives in the same compute domain as the motion planning analytics. This way, they have access to all the raw data coming from the sensors and can build algorithms to react to sensed data in a mode that matches the overall dynamic model of the system. On the other hand, the algorithm integration challenge together with the power and cost required to communicate and process such information centrally increases the cost and power consumption of components.

6.5.5 Resource/Failure Analyses and Dependability Considerations

Communication Bandwidth, CPU Resources, Latency Analysis

Requirements for system assets such as communication bandwidth, processor throughput, memory, latency, etc. can be derived using existing tools and processes. Such analyses will likely show that an IA system will need to have access to nearly every digital resource on the aircraft.

Under Section 6.5.3, we describe a number of ways that an IA system can be integrated into an avionics suite. One concern is being able to access all the points in the avionics suite that the IA system needs for sensing and control. Lack of access is not only spatial but also a temporal concern. That is, without a global network, some data would not be accessible at all and some would have prohibitive latency and jitter. Creating an avionics architecture with sufficient reach, timeliness, and dependability could be very expensive. E.g., it would take 24 AFDX switches to match the bandwidth and dependability of just one ARINC 659 (SAFEbus) backplane, which still would not be all the needed avionics signals.

The need for this amount of AFDX hardware is largely due to the median message size being 8 bytes (dominated by legacy 4-byte 429 messages), for which SAFEbus is 91.4% efficient, but Ethernet is only 9.5% efficient. An additional problem with AFDX is the lack of sufficient VLs to handle all the data an IA would need. Our experience with AFDX on 787 indicates that packing/unpacking smaller

messages into larger Ethernet frames is not a satisfactory solution to these problems. The packing and unpacking operations were very costly in processor resources, at both ends of the communication. This led to the flight control processors running out of throughput.

Failure and Vulnerability Analysis of the Composite Architecture

Accident prevention is generally identified as the primary objective of flight safety programs and policies. The primary goal of accident investigation is to unravel, in a time sequence, all the events which led up to an accident, in order that some action can be taken toward prevention of similar accidents.

The industry has never done a failure modes and effects analysis (FMEA) of a pilot nor a fault tree analysis (FTA) of the pilot/avionics system as a whole. So, there is no history to call on for IA systems replacing a pilot. The question of “how it is different from traditional FTA?” needs to be addressed along with any issues/challenges that arise from the answers.

Furthermore, current assumptions are that “software doesn’t fail”—and thus design failure analysis (or reliability analysis) of the software is not done (in reality, no approach has been posited to do this). DO-178C says “at the time of writing, currently available methods did not provide results in which confidence can be placed.”

The prescriptive guidance of DO-178C provides process level assurance for the correct operation of software according to the system’s design assurance level (DAL). Since human pilots are considered the ultimate arbiter of safety, this approach has worked for current avionic systems.

However, the autonomous role of IA capabilities challenges this notion—i.e., the IA has control authority; therefore, its internal failure modes need to be analyzed. We need to address the question: How do we assess the “functional failure probability” of an IA system (apart from hardware failures)? Or, we need to construct an assurance case argument including elements of prescriptive guidance, verification of safety properties, and run-time monitoring. Assurance methods considerations have been addressed in our previous report of this project [7]. Monitoring considerations are further discussed in Section 6.3.

Additional Dependability Considerations

Based upon the types of functionality and computing architecture (e.g., neural networks) being proposed for IA systems, the impact of internal hardware and design faults needs to be examined from the following perspective:

- Handling SEUs: An IA system probably will have a very large data space (duplication of nearly all non-transient avionics data plus its internal state), making single event upset (SEU) a bigger issue. The topic of resiliency to internal hardware faults in neural networks, however, hasn’t received much attention. Thus, it is difficult to assess how it would be different from traditional processing architectures. Such assessments need to be conducted in order to properly assure dependability of IA systems.

- Potential Redundancy approaches: If an IA capability is used as redundancy for the remaining human pilot, there is a need to enforce separate sensor inputs between them and also separate mechanisms for actuator controls. This is similar to the redundancy approach in current two-pilot systems—providing redundant sensors, displays, and controls in the cockpit. In IA systems, however, the architectural elements and mechanisms to provide such redundancy would be different. For example, IA system architecture may tap into intermediate signals/data within the system, rather than provide a fully redundant path from the sensors to processing to displays. Such situations need to be carefully analyzed as part of the architecture refinement in the context of safety analysis.
- What does dissimilarity mean for IA systems?: Dissimilarity is used at the processor instruction set level and software design level to protect against common-mode failures. In the case of IA capability, providing design dissimilarity raises the following questions that need to be analyzed:
 - Dissimilar AI algorithms?
 - Dissimilar Training data?
 - Dissimilar Objective functions, value functions, rewards?
 - Dissimilar neural network architecture?

6.6 **Autonomy Architecture for Assurance**

Elsewhere in this report, we address the topic of Architectural Considerations for Increasingly Autonomous Systems. That discussion concerns the overall system architecture, but there is a separate, more narrow, topic that concerns the architecture of the autonomy component itself. For example, should it be a monolithic system, or the composition of many smaller components? This choice, and particularly the extent to which subcomponents are independent, will impact assurance and V&V.

When we think of IA aircraft, and reduced crew operations in particular, we generally think of the automation standing in for one of the human pilots. Thus, the tasks to be automated would seem to be those of system management and strategic and emergency decision making, since lower level tasks are already adequately automated by the autopilot, autothrottle, flight management system, and so on. Thus, we might suppose that IA implementation will be some rather complex integration of pre-planned procedures, machine learning, and automated reasoning that mainly operates as a performer of high-level tasks and a manager for classical lower-level automation.

It is interesting, therefore, that one IA system that claims to be able to operate autonomously through severe weather and emergencies has a completely different architecture. The *Intelligent Autopilot System* (IAS) is described in a series of papers by Baomar and Bentley [16–19] and its top-level architecture is one of largely autonomous components coordinated by a very simple flight manager.

The basic approach in IAS uses “apprenticeship learning” to develop automated procedures for elementary tasks by observing a human pilot (in a simulator). It is not entirely clear what form of learning is employed, as much of the discussion concerns the mechanics of neural networks, but [17] mentions inverse reinforcement learning (which is a standard method for this purpose). However, rather than learn some integrated capability, the method is used to learn multiple (initially 4, later as many as 14) small functions. For example, one of these functions takes speed and altitude data and generates gear, brakes, and throttle commands; a second takes roll data and generates aileron commands; a third takes heading data and generates rudder commands; while a fourth takes speed, altitude, and pitch data and generates elevator commands. These functions are all rather simple, and are implemented as neural nets with a single hidden layer containing rather few neurons. Subsequently, comparable functions for take off, rejected takeoff, taxi, fire, cruise throttle, cruise pitch, and emergency landing altitude and pitch are added, followed later by functions for final approach and for landing. These functions are all managed by a simple flow chart (called the *Flight Manager*) that executes several times a second and checks a few predicates and gives control to one or more of the learned functions (e.g., on leaving cruise, the final approach altitude and pitch functions and the rudder and gear functions are all active).

There is no discussion of V&V nor of assurance generally for IAS. But the authors describe some impressive accomplishments in tests. The use of multiple, individually simple, learned functions is attractive, as these should be easier to validate than integrated functions (although we describe caveats below). But, this leaves unaddressed the question of how to provide assurance for their integrated operation. We can easily imagine scenarios where ignorance of overall system state or of the behavior of other functions can lead to inappropriate response (e.g., roll and yaw coordination with an engine out). Other than by experiment, there seems no way to provide assurance for integrated behavior without strong knowledge of the behavior of individual functions. Since the individual functions are relatively simple, such strong knowledge might be feasible: with only a single hidden layer and relatively few neurons. The ANN-optimized SMT capabilities mentioned later in Section 7.2.1 could allow formal analysis of individual functions. However, we have concerns that the individual functions are simple only because they are simplistic. For example, the function for dealing with fire is trained on but a single scenario. And, there is no indication that the elevator controller, for example, considers different flap settings, fuel distribution, gear position, partial hydraulic failure, or other complexities. A fully capable elevator controller will not be simple. It might be argued that, unlike a classical deterministic controller, a controller learned from human behavior will be flexible and adaptable and will cope with complex and novel circumstances. But, there is no assurance of this. The extraordinary safety of current commercial aircraft is built on anticipation of every circumstance and hazard, with assured behavior in every one of those circumstances. We should surely build on this foundation.

If we reject the IAS implementation, what might be a more assurable alternative that retains some of its architecture? Certainly, the idea that IA automation should separate integration from lower-level automation is attractive and consistent with the traditional three-layer robot architecture. But, we believe the lower-level functions should be developed to explicit requirements and assured to satisfy

them, just as they are at present. The integration function is then developed and assured relative to those assured requirements.

For tractability, we suggest that each lower-level function should be largely independent of all others. If component i delivers claim c_i contingent on satisfaction of assumptions a_i , then i is independent of component j if a_i is logically independent of c_j . Thus, we might expect that the radios are independent of the autopilot. Of course, our knowledge is often incomplete and an expectation of independence is only to the best of our knowledge.⁹ If a_i does depend on c_j but a_j does not depend on c_i , then we have a hierarchical arrangement, such as that between the flight management system and the autopilot (the former depends on/uses the latter but not vice versa). If components i and j are mutually dependent (as might be the case between the yaw damper and the roll components of the autopilot), then we need either compositional or integrated assurance. Of course, components can fail or enter degraded modes. So, in a prior report on modular certification [107], we suggested that claims and assumptions should be “stepped” to avoid cascading or “domino” failures. Furthermore, we suggest that each lower-level component should be enabled throughout the flight and should use local sensor input and situation awareness to determine whether it should be active rather than passive and, if the former, what it should do.

In summary, we suggest that automation should comprise continuously enabled, largely independent or hierarchical components, each provided with explicit requirements (i.e., claims and assumptions) and assurance that these are satisfied. The expectation for requirements and assurance is the same whether a component is conventional, or has autonomy capabilities that may be implemented by machine learning and ANNs.

With this architecture, we suggest that the top-level function (that integrates all the other components) can be rather simple, amounting to little more than a sequencer and source of trustworthy situation assessment. The latter point is important: for example, there is speculation that in the crash of Emirates flight 521 during an aborted landing on 3 August 2016, the landing gear retracted because it assessed the aircraft to be airborne, while TOGA was inhibited because it assessed the aircraft to be on the ground, resulting in a wheels-up crash landing. Requiring the top-level IA function to be a source of situation assessment is consistent with the idea that IA should be capable of many tasks currently performed by humans, where the pilot is “the integrator of last resort”. Of course, the automation of an IA aircraft must perform many functions in addition to integrated flight control: e.g., fault diagnosis and reconfiguration, route planning and navigation (e.g., for weather), ground taxi, radio communications, emergency procedures, and many more. As IA systems take on larger responsibilities than current aircraft automation, there will larger areas that require integrated situation assessment and coherent behavior: for example, if the radios announce that we are diverting to Omaha, then we need to be sure that navigation is taking us to Omaha.

⁹This statement should not be taken as condoning wallowing in ignorance. It is the responsibility of all safety-critical designers to be as knowledgeable as possible, including understanding rare failure modes [42].

The relationship between a lower-level component and the integrator may be hierarchical (necessarily, the integrator depends on all the lower-level components) or one of mutual dependence (e.g., if a component depends on the integrator for situation assessment). There may be a choice whether to rely on local sensing or integrated situation assessment and this will affect dependence. For example, pitch control may respond to the drag of gear down either because it has access to the gear state (from the integrator), or because its aerodynamic sensing and control just deals with it. In cases where multiple components make discrete choices based on discrete situations (as in the gear retraction and TOGA example), integrated situation assessment seems best, as it enables coordinated and coherent response. On the other hand, continuous control may be best handled by local sensing. For example, locally sensed pitch control should cope with asymmetric gear deployment (one up, one down), while there seems no good output for integrated situation assessment in this case.

Emergencies pose another interesting conundrum for the relationship between lower-level automation and the integrator. For example, in the absence of air data, current autopilots and autothrottles will typically disconnect and abruptly hand control to the pilots (the Air France 447 scenario). One design choice is for the integrator to search for another way to maintain the goal of safe flight and to conclude that it must itself take command and maintain the previous pitch and thrust,¹⁰ while another is for the autopilot and autothrottle to do this and not to disconnect. In all cases, the aircraft is in an uncertified and uncertifiable (and unsafe) condition. It seems consistent with the idea that lower-level components operate continuously and “never give up” that the autopilot and autothrottle should perform this function, while also signaling that they are in uncertified circumstances. The integrator can search for other solutions and can disconnect the autopilot and autothrottle if it does find a safer alternative.

As the AF 447 emergency scenario suggests, the integrator must be more than the sequencer proposed earlier. Rather, it is essentially a planning and constraint satisfaction engine: planning to advance its goals (reach the destination), and constraint satisfaction to do so safely (given the requirements specifications of the available components, and the current situation assessment). It is a topic for investigation whether these functions need to be performed online, or can be precomputed, thus restoring the integrator to a sequencer. In all cases, the integrator must be able to explain its choices to a human pilot and suggest consequences and alternatives.

¹⁰This is the recommended action for human pilots.

7 V&V Issues and Approaches

A key issue for autonomous systems is rigorously verifying and validating their safety. As we have mentioned earlier in Section 2.4, verification and validation (V&V) for current avionic systems is guided by well-established guidelines (such as [104]), techniques and tools to aid engineers verify and validate whether the system is safe. However, the adaptive, flexible, context-sensitive behavior of autonomous systems causes unpredictability and emergent behavior—that was not necessarily specified, intended or consciously implemented during the system design. Hence, new V&V techniques are warranted to address the safety assurance challenges of such systems.

In this section, we first describe some exemplars of V&V challenges posed by IA systems; then we explore the current capabilities of a broad range of V&V approaches such as verification, simulation, run-time monitoring and testing currently explored for such systems.

V&V Challenges in IA Systems

In Section 2.3, we discussed challenges for IA systems' safety assurance that also impact V&V. While verifying a system observes inputs or sets values correctly is basic V&V, the challenges with IA systems lie in verifying the functions of IA systems that involve complex decision making, troubleshooting, and maintaining consistency between the pilot's and system's understanding of the situation. Here, we present some example scenarios arising from those challenges:

- *Correct response to an incorrect command:* In several tasks, the co-pilot is expected to query the pilot or ATC if there is an unexpected command. The query should explain why the command is unexpected, informing the pilot of the system's understanding of the situation. Off hand, we don't know how often the command is in error or the system is misreading the aircraft state or environment. For an IA system, a simple override, to do as originally instructed, may not be sufficient. The system would be left in an uncertain state, performing actions that do not follow from the synthesis of data it is taking in.
- *Dynamic decisions about keeping the pilot informed but not overloaded:* The design and rollout of cockpit systems involves an exhaustive study of pilot interaction. How much information is enough/too much? At what point is an alert a distraction making an appropriate response more difficult? An IA system may need to decide in the moment what information is most needed or distracting. What if it replaces data the pilot needs with information it thinks is relevant?
- *Parameters for continuing operation when failures occur:* Sensor loss or mechanical malfunction that can be handled safely by experienced pilots would leave an IA system operating outside traditional safety envelopes. An IA system needs to make a best effort, may be with reduced confidence, rather than drop everything on the pilot. The verification challenge lies in the sheer number of sensors and subsystems that may fail in any number of combinations as well as the calculation and communication of confidence in the situational awareness given loss of information or control.

To support such autonomous behaviors and decision making capabilities, IA systems will employ unconventional implementation techniques such as machine learning (Section 2.3.4). As we have noted in Section 2.1, V&V issues have been reported in several safety critical domains that use machine learning. Therefore, V&V methods and tools focusing on machine learning implementations need to be developed.

7.1 Issues in V&V of Machine Learning Techniques

IA systems will increasingly employ functions developed using various machine learning techniques. Machine learning is a general term that encompasses a broad class of algorithms that learn from data and adapt to new situations while in operation based on the learning so that machine can perform the “cognitive” functions like a human. Rapid advances in ML based techniques has vastly improved the capabilities of such systems. While advanced learning technologies are essential for enabling coordination and interaction between the autonomous agents and the environment, a careful analysis of their decision reliability in the face of carefully crafted adversarial samples and thwarting their vulnerabilities are still in their infancy.

We divide our discussion into three sections, corresponding to the traditional subfields of machine learning: unsupervised, supervised, and reinforcement learning. *Unsupervised learning* is applied to tasks such as clustering and anomaly detection; it is based on discerning structure or patterns within the training data. Unsupervised learning may not find much application to control tasks in IA aircraft, but we think it can be useful in adjuncts that perform monitoring and runtime assurance for these tasks. In *supervised learning*, each input has a numerical value or discrete label associated with it and the task is to predict these, given a training set of correctly labeled examples. Supervised learning is often used for image classification (e.g., to recognize road signs in automated driving). Aircraft applications might include failure diagnosis and prognosis (e.g., early warning of engine failure by integrated analysis of multiple sensor inputs). *Reinforcement learning* is somewhat similar to adaptive control: the learning agent perceives the state of its environment (e.g., a plant it is intended to control), takes actions accordingly, and receives a reward following each action; the task is to learn a strategy for actions that maximizes rewards over the long term. A variant method is used in learning from human experts.

7.1.1 V&V Issues in Unsupervised Learning

Unsupervised learning is used to look for patterns or groupings in data. The learning algorithm is typically specialized to the kind of pattern that is expected. Applications of unsupervised learning include clustering and anomaly detection, as portrayed, rather impressionistically, on the left and right of Figure 20, respectively.

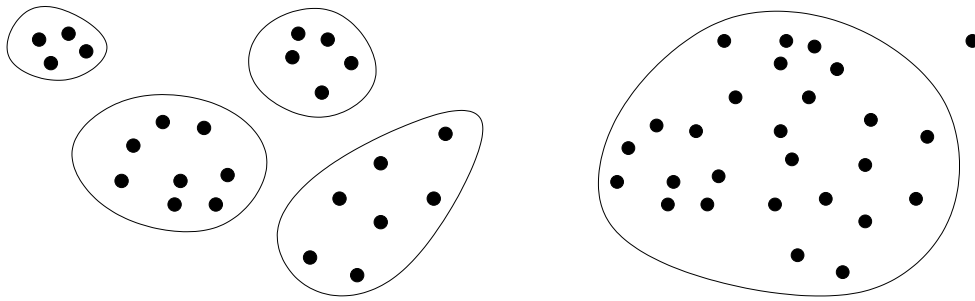


Figure 20: Unsupervised Learning for Clustering and Anomaly Detection

A correctness specification for unsupervised learning could comprise a “ground truth” assignment for each input, i.e., the cluster or group it should belong to—although this is somewhat contrary to the purpose of unsupervised learning, which is to *discover* such groupings. Consequently, “ground truth” assignments are seldom available for real applications, so we may need to infer performance of the chosen learning algorithm from controlled experiments on data for similar applications. There are, however, some weak measures that do not require ground truth, such as the *silhouette coefficient*, which indicates the “coherence” of each cluster.

There is an interesting application for unsupervised learning where we do have a “ground truth” (although we will weaken this later) and this is a form of anomaly detection where we wish to compare an input encountered at runtime against a training set that consists entirely of “normal” inputs. The intent is to determine whether new inputs are normal or anomalous [61].

Although anomaly detection is often understood as a topic in unsupervised learning, it also arises in control theory, where somewhat different techniques have been developed. Here, the idea is to learn some “envelope” that surrounds an established set of data and then compare new inputs to see if they are inside or outside the envelope. An example of such an envelope might be one for sensor data encountered during testing; then, at runtime, we can tell whether or not we are within the tested space. Of course, the distribution of inputs within the envelope could be sparse, so being inside the envelope might not guarantee we are “close” to a tested input, but it suggests a promising approach that can be tuned and improved.

One way to tighten the envelope is introduce “modes”: we might expect there is a subenvelope of inputs where the airplane is in level flight that is different from that where it is climbing or descending and different again from that where it is turning. Figure 21 portrays such a “moded” envelope (solid line), which may be compared with a conventional one (dotted line) that misclassifies the lone point.

Tiwari and colleagues [134] present an algorithm for learning envelopes with modes that is simple and effective; note that we do not have “ground truth” for modes: for each input in the training set that is outside the current envelope, the algorithm must decide whether to extend some existing mode, or to create a new one. The algorithm is always correct, in that the envelope truly encloses the training set, but there are no guarantees how tight it is, nor whether the training set was sufficiently large and representative to cover all intended behaviors, nor whether the discovered modes truly correspond

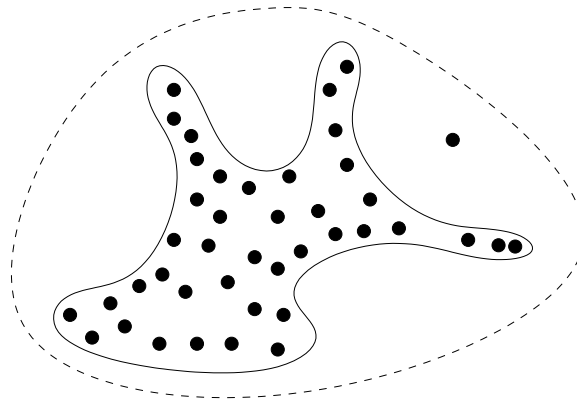


Figure 21: Envelope with Modes

to those in the system whose behavior is observed; empirical results are very encouraging, however. Observe that in computer science terms, an envelope is an invariant, and a moded envelope is a disjunctive invariant [106].

The motivation for envelope construction was detection of spoofing attacks on sensors [123, 134], a classical application of anomaly detection. However, we will later suggest its application to runtime verification for machine learning.

7.1.2 V&V Issues in Supervised Learning

Supervised learning differs from unsupervised in that every input in the training set has a numerical value (“regression learning”) or a discrete label (“classification learning”) associated with it. The task is to learn and generalize from a correctly labeled training set in order to predict the values or labels of inputs presented at runtime.

Supervised learning methods first select a *feature set* (i.e., some attributes derived from the raw inputs)¹¹ and a set of *candidate functions* (e.g., those that can be represented by a 10 layer neural net) to encode the learned behavior,¹² and then iteratively optimize the function (e.g., by adjusting weights in the neural net) as training inputs are consumed. Topics of concern include the size and representativeness of the training set, the choice of feature set and class of candidate functions, and the optimization strategy. For example, if we wish to recognize pictures of cats, we might first preprocess the images (e.g., using edge detection) in a way that highlights features such as pointed ears and whiskers. One of the advantages claimed for deep neural nets (i.e., those with hundreds of layers) is that they are less dependent on (human guided) feature selection as they do it themselves; this is controversial and there is much current debate on understanding and explaining deep neural nets (see later for references).

¹¹This is true of all learning methods: they may work better if we extract and present salient features from the inputs, rather than the raw inputs themselves.

¹²Although deep neural nets are now popular for supervised learning, there are other methods, such as Support Vector Machines.

The optimization strategy must steer a course between *overfitting* (i.e., adapting so closely to the training set that it fails to generalize and therefore performs poorly on new examples) and *underfitting*, which leads to poor performance for the opposite reasons. One criticism of machine learning methods for safety critical systems is that optimization procedures generally consider just the “distance” between predicted and actual values, where some measure of the (safety) *cost* of misclassification might be more suitable [48, 136].

The size and representativeness of the training set is a concern for all forms of machine learning. Superficially similar training sets can lead to widely different performance (e.g., 58% vs 92% accuracy [38]) and there are no strong guarantees on when a training set is “big enough”. Commercial airplanes are certified for, and achieve, extraordinary levels of safety, which means that very rare “1 in a billion” circumstances must be considered seriously [42] and it is difficult to imagine that feasible training sets will contain examples of this rarity. Recent accidents involving self-driving cars suggest they were not trained on, and did not recognize, moderately rare scenarios.

7.1.3 V&V Issues in Reinforcement Learning

Reinforcement learning is about training an agent to interact with its environment in a way that maximizes long term performance. The agent repeatedly receives inputs describing some aspects of its environment, performs an *action*, and receives a *reward* (which may be negative—a form of punishment). This is performed in the context of some model for the environment, which is often a Markov Decision Process (MDP)¹³: that is, a nondeterministic state machine where actions determine the transitions available from the current state, and each of those transitions has an associated probability (summing to 1) and a reward. The states correspond to some partitioning of the input data, and rewards represent performance (e.g., a measure of the “safety” of the resulting state). In *model-based* reinforcement learning, the states and transitions are known or assumed *a priori* (but not their probabilities and rewards), whereas these are not known in *model-free* learning.

The agent learns a *policy* that is a function from states to actions that maximizes long term rewards. There is a tension while learning between selecting the best known action in the current state, and exploring others for the purpose of gaining information (it may be that a sequence of low reward transitions ultimately leads to a high payoff). Typically, the best known action is performed with probability $1 - \epsilon$ and another is randomly chosen with probability ϵ . In some contexts, learning is performed offline, prior to deployment, while in others it continues post deployment (online learning). In the latter case, we generally need to be careful that exploration does not lead to unsafe states.

There are several variations on reinforcement learning: *inverse* reinforcement learning observes expert performance and aims to learn the reward function, while data mining can use reinforcement learning to reconstruct an agent’s behavior. For example, researchers at NASA AMES use this form

¹³Discrete Time Markov Chains are another popular choice.

of data mining to discover the precursors to over- and underspeed events in logs from event recorders of commercial airplanes [73–75].

Reinforcement learning raises most of the standard questions about machine learning: how do we know if the training set was sufficiently large and representative, and how do we know that the learned policy has generalized effectively from the training set? However, unlike most applications of supervised learning, it is often feasible to guard a reinforcement learned system with a safety monitor that can block unsafe actions. This could be particularly useful with online learners, although the safety guard may interfere with the optimization method [57, 135]. An alternative is to evaluate the safety of actions during offline learning using formal verification or evaluations and to adjust the model or data appropriately [60].

Reward hacking is a concern that is specific to reinforcement learning; it arises when high rewards can be obtained in a way other than the developers intended. For example, a robot vacuum cleaner that is required to clean until its sensors detect no further dirt might simply turn off its sensors. Amodei and colleagues claim this is a common problem with complex systems and suggest some countermeasures [11].

7.2 V&V Approaches for Specific Machine Learning Implementations

Based on the challenges and issues outlined in Section 2, we further explore current and upcoming approaches for developing assured ML systems as well as verifying specific types of ML implementations. We also identify issues and further challenges with these approaches.

7.2.1 V&V Approaches in Artificial Neural Networks

“Artificial Neural Nets” (ANNs) are widely employed in machine learning, and so their properties and V&V issues are worthy of separate study. A generic ANN is portrayed in Figure 22. In general, the net consists of input and output layers, and some number of intermediate (aka. hidden) layers. Until recently there were few (typically 1) intermediate layers, but modern, so-called “deep”, ANNs may have hundreds. Each layer consist of many individual “Artificial Neurons” (ANs), each receiving inputs from (all) the neurons in the previous layer and feeding its (single) output value to (all) the neurons of the succeeding layer.

All inputs and outputs are real-valued numbers; each AN has a real-valued *weight* associated with each of its inputs (there are variants where inputs also have *biases*) and computes its output as the application of some *activation function* to the weighted sum of its inputs. The values of the weights are determined by some learning procedure applied to a set of training data; If an AN has n inputs i_1, \dots, i_n with weights w_1, \dots, w_n , then its output is $f(\sum_{j=1}^n w_j \times i_j)$, where f is the activation function.

The purpose of the activation function is to enable switching behavior, so it is usually some strongly nonlinear function such as arctangent, or (more usually) the *Rectified Linear Unit* (ReLU), which simply replaces all negative values by 0.

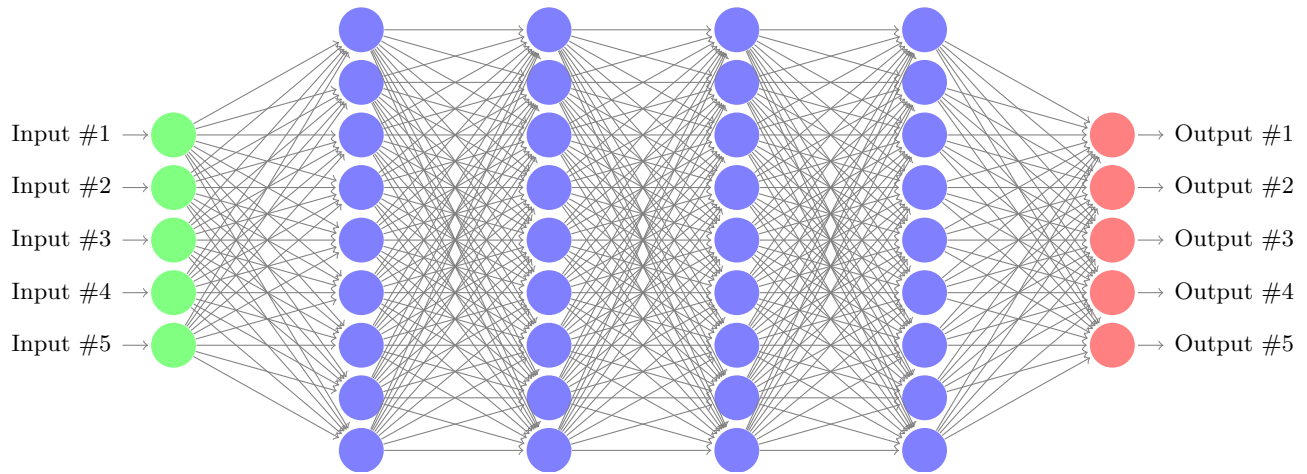


Figure 22: Artificial Neurons and Artificial Neural Net

The relation between inputs and outputs of a full ANN is the composition of many AN calculations of the kind described. If we have a specification of properties required of this relation, then its verification is a problem in linear real arithmetic and can, in principle be solved by a modern SMT solver. A difficulty in practice is that each ReLU function invokes a case split and there are exponentially many of them, leading to infeasible solution times. However, there are now several optimized methods for dealing with this class of problems, so that verification of properties of large ANNs is quite practical [30, 43, 80].

Unfortunately, there are rather few applications where we have an independent specification for the function computed by an ANN: the collision avoidance algorithm ACAS Xu is a rare exception (and consequently receives much attention) [79]. More commonly, we have a set of examples and these are managed or partitioned in some way and the ANN is trained (i.e., its weights are tuned) on one group of examples and evaluated on the other. Typical applications are those of machine learning, as described in the previous sections.

It is not uncommon for the evaluation to indicate extremely high accuracy, but we have to ask whether this really does provide adequate assurance. One immediate concern is whether the training and evaluation data are really representative of the examples encountered in real life. One promising idea is separately to learn a compact representation of the inputs to the training set. This is “envelope construction”, which was discussed as an application of unsupervised learning (recall Section 7.1.1). An envelope would allow runtime determination whether the current inputs are “inside” or “outside” the space of examples encountered in training. There is necessarily some error with edge cases and this can be tuned to err on whichever side is preferred. As noted earlier, use of “modes” can tighten the envelope [134].

This approach has not yet been evaluated in practice, but there are indications of potential difficulties. One concern is the existence of *adversarial examples* for classification problems. These are inputs constructed by perturbing a given input (typically an image); to a human the original and perturbed

inputs seem indistinguishable and both are classified similarly, yet the ANN classifies the perturbed example completely differently than the original: e.g., a perturbed roadside stop sign is classified as a birdcage. Since adversarial perturbations can be very small (just a few pixels), it is plausible that the runtime check will consider them “inside” the training set and thereby raise no warning.

The existence of adversarial examples raises a more general concern: it indicates that the ANN might be responding to completely different input features than do humans, and that rather than constructing human-like generalizations from the training data, it has merely memorized exactly what it has seen (or inscrutable aspects of this, such as the colors of certain pixels) [142]. There is much current work that seeks to understand and characterize how generalization and learning are actually accomplished in deep ANNs [82, 124], but there is no consensus as yet.

Although adversarial examples are just that—constructed by “adversaries”—they raise the concern that classifications learned by ANNs may not be robust, and that naturally arising inputs different from the training set, yet close to it, may be classified incorrectly. Since we do not really know how ANNs work, we cannot build a fully principled defense against this (or, dually, verification of its absence), but there are some promising ideas.

One approach is a form of anomaly detection, which uses a separate unsupervised learning procedure to detect inputs that might be adversarial [96]—it is not known whether this might work also for “natural but difficult” examples but it seems plausible. The intuition behind this and related emerging methods is that one expects a reasonably smooth “geometry” in the input space and these methods try to construct or detect such smooth “manifolds” [103]. Observe there is some similarity to the “inside-outside” detector based on envelopes mentioned earlier. A variant is to use SMT solving to guarantee that no adversarial examples exist within some small distance of training examples or other selected regions [71].

Response to V&V Issues

Unconventional implementations, such as machine learning with ANNs, present a challenge to conventional methods of verification and validation. Nonetheless, it is possible to seek some interpretation of conventional methods for these implementations (e.g., for ANNs [29]). Even if the learning or ANN component does not lend itself to a detailed requirements specification, the avionic function of which it is a part should do so, and this should enable some conventional methods of V&V, such as requirements-based testing (although there are suggestions that Combinatorial Testing, aka. High Throughput Testing, may provide more appropriate coverage metrics than MC/DC in this context [126]). The training data used for learning-based systems could be considered to constitute aspects of a “design” and it would be worthwhile to develop methods for evaluating these data, particularly for some notion of “coverage”.

Beyond conventional V&V, there are conventional methods of fault tolerance that may be effective as a means of assurance, such as various forms of diversity. For example, we could employ a variant on “N-Version” software [15], where multiple “diverse” machine-learning implementations (e.g., trained

on different data, or on the same data but by different methods) are voted or required to agree. It is possible that non-faulty learners will deliver different results for the same inputs, so the notion of voting or agreement must be suitably flexible. An alternative to software diversity is data diversity [9]. This arises naturally in sampled data, where temporal continuity may be a reasonable expectation. For example, if a sequence of inputs from the camera of a moving car classify an object a stop sign, and then one classifies it as a birdcage, we might continue to use “stop sign” as the preferred interpretation until or unless the “birdcage” classification stabilizes.

Temporal continuity is related to the expectation that outputs should be relatively stable: obviously the output of a classifier will change periodically from one classification to another, but we expect these changes to be orderly and not erratic. It seems feasible to construct “runtime enforcers” for stability using the ideas of anomaly detectors and envelopes that were discussed (as defense for adversarial perturbations) in the previous section.

It is worth noting that most of the assurance and V&V methods suggested here are runtime methods. This is a natural consequence of the general absence of design-time documentation for learning-based systems. However, the runtime methods we suggest for individual functions focus on aspects of their execution behavior (e.g., erratic classifications), and this is different to the safety-based pervasive monitoring that we advocate for overall system assurance.

7.2.2 Safe Reinforcement Learning

Although successfully used in various applications, traditional RL is not applicable to problems where the policies synthesised by the agent must satisfy strict constraints associated with the safety, reliability, performance and other critical aspects of the problem. Given its relevance in industrial applications, there has been a growing interest in safe learning methods in the recent years.

Safe RL can be defined as the process of learning policies that maximize the expectation of the return in problems in which it is important to ensure reasonable system performance and/or respect safety constraints during the learning and/or deployment processes. An exploration process is considered safe if no undesirable states are ever visited, which can only be achieved through the incorporation of external knowledge.

The most straightforward way of guiding a “safe” exploration is to provide supervision or teacher. For example, in a Q-learning setting, given a teacher (usually a human) to provide advice, the agent acts on the teacher’s advice that is provided whenever the teacher considers it to be necessary to prevent catastrophic situations. Otherwise, the agent chooses randomly between the set of actions with the highest Q-values. At each time step, the teacher can tune the reward signal before sending it to the agent. While this is considered the “safest” learning approach, having a human teacher direct the agent in thousands of learning episodes is nearly impossible. Extensions of existing RL methods are being explored by various researchers and practitioners to enable Safe Reinforcement Learning. While a comprehensive survey of safe RL approaches are detailed in Garcia et.al [57], we discuss some of the most relevant ones here.

To that end, Alshiekh et.al [6] introduced a *shield* into the traditional RL setting. The shield is computed upfront from a given set of safety specifications and an abstraction of the agent’s environment dynamics. The shield restricts the agent as little as possible and forbids actions only if they could endanger safe system behavior. This approach modifies the loop between the learning agent and its environment in two alternative ways, depending on the location at which the shield is implemented. In the first one, the shield is implemented before the learning agent and acts each time the learning agent is to make a decision and provides a list of safe actions. This list restricts the choices for the learner. The shield allows the agent to follow any policy as long as it is safe. In the alternative implementation of the shield, it monitors the actions selected by the learning agent and corrects them if and only if the chosen action is unsafe. In both ways the shield acts like the “teacher” who directs and corrects the learning as per the safety specifications.

Further, Mason et.al [90] proposed an assured RL approach that restricts the exploration of the RL agent that are guaranteed to yield solutions compliant with the required constraints. In the approach the uncertain environment is modeled as a high-level, abstract Markov decision process (AMDP), and use probabilistic model checking to establish AMDP policies that satisfy a set of constraints defined in probabilistic temporal logic. These formally verified abstract policies are then used to restrict the RL agent’s exploration of the solution space so as to avoid constraint violations.

A recent work by Fulton et.al [56], formal verification combined with verified runtime monitoring was employed to ensure the safety of a learning agent. Verification results are preserved whenever learning agents limit exploration within the confounds of verified control choices as long as observed reality comports with the model used for off-line verification. When a model violation is detected, the agent abandons efficiency and instead attempts to learn a control strategy that guides the agent to a modeled portion of the state space.

Alternatively, some form of risk measure is used to quantify the safety value of an action. This risk measure is combined with the action value to form the risk-adjusted utility of the action [84]. For instance, Gehring et.al [58] define a risk metric based on controllability, where a controllable state is defined as a state where the effects of an action are easy to predict. Further, Moldovan et.al [97], introduce a criteria to avoid irreversible actions by entering only states from which the agent can return to the initial state. Risk direct exploration introduces a safety bias in the exploration process but does not satisfy hard constraints.

Despite tremorous work in this area, some challenges still remain. One of the basic questions is how to avoid the agent making unsafe decisions when given inputs that are potentially very different than the data previously seen in the training thus far. There is currently lack of techniques to determine training data adequacy and measure its coverage with respect to the real environment the agent is going to operate. Broadly speaking, any agent that is not trained on a given situation may misunderstand it and potentially commit harmful actions. Furthermore, the agent might not even be able to recognize such actions as harmful. In the reduced crew operational scenario, though one can generally assume that the human pilot could guide the IA system if it encounters an unexplored scenario in most cases,

when the pilot is incapacitated or emergency situations. As such, a key skill for a safer agent is to be able to recognise its own ignorance, trying to avoid silent failures. While work such as anomaly detection [87] can contribute to detection of such scenarios, this is an open research area.

Further, there is a lack of techniques to access correctness and adequacy of the notion of safety in agents. While techniques to formally guarantees safety properties are being explored, they are still in their inception phase. They need to be matured for scalability to handle real world scenarios. Further, the reward hypothesis and Objective functions, being the premise for the optimal performance, should the they be ill-posed, the policy devised could have hazardous results. There needs to be safety bounds to prevent such situations.

7.3 Compositional Verification Challenges

Compositional verification (CV) is a well known technique to formally verifying large systems by decomposing the verification task into smaller components. However, this technique relies on composable *contracts* or component-level specifications that can be composed to prove system-level specification. Assume-guarantee reasoning [78] provide approaches to perform compositional verification in a principled manner. This has been extensively used in several safety critical application development that use formal methods for verification. However, machine learning and AI based systems pose unique challenges for compositional specification and verification.

Behaviorial Specification: While CV heavily relies on having a precise, mathematical description of the component and system behavior, specifying them for autonomous vehicles is not straightforward. The current techniques to used to specify functional properties are not adequate to capture the behavior of the dynamic, non-linear components with stochastic behavior based on thousands of parameters. Further, as learning becomes sophisticated, there will be more and more emergent behaviors that arising as a result of complex interactions among components. Even further, as the system learns, the behaviors evolve and there will be stochasticity in the behavior. Hence, there needs to be effective methods to identify and quantitatively specify desired and undesired properties of such components and systems in a way that can be incrementally updated. Work on parameterizing contracts [20] and devising abstractions to specify properties, though preliminary, are promising directions in this area [120].

Verification Engines: Tooling (hardware and software) is an essential part of employing formal verification. While there are advances in the solver technology such as SAT and SMT, as well as model checking tools such as PRISM [90] have been shown to verify learning systems, their scalability and capability to verify the adaptive behavior of AI-based systems is still at its infancy. Rather than verifying the entire state space of systems, newer, scalable techniques that analyze by pruning and/or abstracting the state space are much needed.

Environment Determination: Precise determination of the system-environment interface is crucial for verification. However to verify learning based systems one needs to first define all possi-

ble features (variables) in a non-deterministic, stochastic environment, that turns out to be a combinatorial explosion problem. Also, being able to filter out infeasible scenarios in the environment while performing verification is important; otherwise, the verification will endlessly report spurious violations (counter-examples that show the scenario that lead to the violation). In system involving joint human-machine control, such as reduced crew operations, being able to specify behaviors of human are a key part of the environment specification. However, the variability and uncertainty in human behavior makes this task arduous. However, recent work on automatically deriving the weakest required environment [32,50] from system properties, as well as work on capturing mental model of humans [54] are directed towards addressing these challenges.

7.4 Challenges in Testing of IA Systems

While there is a consensus that in general testing of complex systems is hard, it is obvious that testing autonomous systems is even harder. In fact, it is still an unsolved area. Traditional scenario based testing methods fall short of providing the desired confidence level, mainly due to the combinatorial explosion of possible situations to be tested in autonomous systems. The non-determinism and complexity of the input space and the context in which the autonomous system will operate makes it hard to even determine the adequacy of test cases. Further, the complexity of the program logic, which is caused by characteristics that are inherent in autonomous systems such as assembling their own course of actions for a given goal, adapting to different environments, learning, diagnosing themselves and reconfiguring themselves to maintain their functionality makes it methodologically and conceptually challenging to assess the coverage of testing on such systems. In fact, a major concern on testing autonomous systems is that the methods are built on the assumption that if a system passes all tests then it will work flawlessly during operation. However, since the behavior of autonomous systems can change over time, it casts doubt on reliability of test results.

Fortunately, there is active research in this area. Nguyen et.al [101] propose specify an evolutionary testing approach to test autonomous agents. The approach is to transform the stakeholder requirements into quality functions that are in-turn used as evolutionary optimization criteria to generate demanding test cases. This enables to test the agent in a range of contexts, including those in which it is most vulnerable to poor performance.

On the other end, approaches such as adaptive testing [118] involves subjecting the autonomous control system to an adaptively chosen set of fault scenarios within a high-fidelity vehicle simulator, and automating the search for combinations of faults over time that drive the autonomous control (and consequently the autonomous vehicle or system) to unsafe or failed behavior. Since this method is simulation based, safety and trust in the autonomous system may be gained by allowing the system to be stressed prior to live environmental testing; Because it is simulation based, many iterations of missions may be tested for failure conditions and mission effectiveness in less time.

Further, Berger et.al [24] demonstrate a complete virtual test environment that could supply the control software with virtual raw sensor inputs to enable automatic testing in the virtual test environment before they are conducted in the real world. While similar approaches have also proposed by [66], the scope of virtual testing is limited given the randomness in the environment and system behavior. Hence, virtual test environments need to represent the intended operating environment in a more systematic fashion. In sum, testing autonomous systems is an area of research that needs lots of work before it can mature.

7.5 Challenges and Approaches for Simulation of Human-IA Interactions

In developing the assurance case strategy for system safety, we must consider the IA system as a primary or secondary control element to fully understand the risks and causal factors inherent in the system. The IA system will not behave as a human, and just as all human pilots have individual mental models, so too will the IA's mental model differ from that of a human, and in some cases diverge from that of a human crewmate.

Best practices in design of Human-IA Interaction Listed below are some examples of system design considerations which would serve to mitigate the human factors risks we have discussed.

- The design of cooperative human-IA systems must induce a shared understanding of state information. Proper CRM should ensure that pilots understand aircraft state and status, task state and status, status of queued tasks and the prioritization strategy.
- Pilots should be kept in the loop as much as reasonable given the complexity of the automation.
- Intelligent assistants which monitor pilot behavior and infer operational intent can detect possible errors in the crew's mental model and provide aid to understanding and properly addressing the situation.
- Crew task allocation must consider a number of factors pertaining to the crew member, including role, resources and time available, workload, experience and acceptance.
- Computational tools may be effective for aiding crew task allocation. Future research should focus on developing algorithms and sensors (including human physiological measures) to measure system state and assess crew workload.
- When tasks are allocated/re-allocated, providing feedback and visual cues allows crew members to identify what needs to be done while minimizing the amount of information that has to be recalled.
- Communication protocols between the human crewmate and the IA should be standardized as much as possible. Language models should support natural language and proper industry vernacular, and should be regionally adaptable.

Simulation should address the effectiveness of IA design to minimize the potential for communication errors, cognitive mismatches, and breakdowns in task management, and to promote shared situation awareness and effective CRM to ensure safe human-IA operations.

Note: To partly address some of the V&V challenges of Human-IA Interaction, Ahrenbach and Goodloe have recently proposed the use of agent safety logic [5]—a mechanized modal logic approach to modeling and analysis of some aspects of pilot-automation interaction. A critical discussion of this approach and suggestion for further investigation is discussed in Appendix A.

7.5.1 Initial and Periodic Knowledge Checks

It must be considered that, in order to understand the IA’s capability as a “pilot”, the system must first be ‘checked out’ similarly to the human pilot, in that it must be subjected to a variety of nominal and off-nominal conditions which will test its rule-based knowledge as well as its aptitude for problem-solving and decision making when the pilot manual alone is not enough to effectively respond to the situation. In addition to this, we must test the capability of the system to engage in and apply machine learning techniques in the same way or more efficiently than the human pilot applies experiential knowledge to a situation.

Simulation and modeling are key approaches to assessing the capabilities of the system, understanding the similarities and differences between human and machine training/learning, and mitigating associated risks. We first need to define the properties of the simulation software to analyze various system behaviors. The simulation must adequately (to X level of reliability) determine the IA performed action(s) (within X time/aircraft performance constraints), to an X degree of accuracy, with an X degree of proficiency, which produced desired result(s) and final outcome(s). For non-deterministic systems, it is usually inappropriate to score all performance assessments as “pass” or “fail”. There may be several gradations of “desired” results and outcomes, such that the scoring of IA performance assessments is never binary, and not always linear. Examples of success factors may include:

- How quickly and accurately was the maneuver performed?
- Did the aircraft stay within its normal operating limits? Did it stay within its threshold limits?
- Was there any damage to the aircraft?
- Did the IA action directly contribute to aircraft damage and/or human injury?
- Could a different IA action have prevented aircraft damage and/or human injury?

The simulation results can also be compared to the simulator performance assessments of human operators to check that the IA is performing at the same or better capacity, as well as to better understand the similarities and differences in the human vs IA operators’ approaches to various situations. This understanding will provide insight into different ways of information processing and decision making; learned differences in behavior will give IA system developers more clarity into what information, expectations and intentions need to be communicated between the crew.

One example of a knowledge test procedure for the human pilot is programming the Flight Management System (FMS). The structure and interaction method for every FMS system (much like every aircraft platform and configuration) may be different, and thus a human pilot may be subject to knowledge testing on a specific system after training, when there is a system update or retrofit, or periodically as part of a general flight review. Through the use of simulated knowledge tests, the IA system can be evaluated in somewhat similar ways. The scoring for each actor, human or IA, should be a hybrid of Pass/Fail and Scaled assessments. Example tasks, expected actions, and knowledge assessment factors are described below:

Demonstration Task 1

The pilot must check that the current software version and navigation database is loaded immediately upon start up.

- Human pilot – Navigate to Maintenance page and confirm software version. Navigate to Nav Ident page and confirm the current Nav DB version. (Knowledge of current versions; knowledge of where this information can be found; what to do if there is a newer version available)
- IA system – Perform automatic software checks at startup. (Knowledge of current versions; successful initiation of checks; successful completion of checks; successful performance of software updates).

Demonstration Task 2

Each operating mode has its own purpose. The pilot must demonstrate understanding of each mode.

- Human pilot – Describe the differences between Dual/Synchronous, Initiated Transfer, and Independent modes. (Mode awareness; knowledge of expected display behavior)
- IA system – Perform a check that automatic mode select corresponds to desired display state. (Successful initiation of checks; successful completion of checks; knowledge of when to reset modes).

Demonstration Task 3

Perform a check for problems with sensors.

- Human pilot – Navigate to Maintenance or Status page to confirm sensors are operating normally. Check FMS status readout against other avionics display information. (Knowledge of where this information can be found; how to troubleshoot problems)
- IA system – Continuously monitor sensor systems and connectivity to FMS. (Proper and continuous monitoring of sensors; how to alert crew and troubleshoot problems).

Demonstration Task 4

Perform a check for correct Heading and Position data.

- Human pilot - Toggle between MAG and TRUE settings and visually confirm the displays change accordingly. Enter origin. Enumerate different methods of Pos Init. (Knowledge of

how to set heading modes and where heading mode information can be found; knowledge of where, how and when to enter and check position information).

- IA system – Perform a check of MAG and TRUE heading displays. Perform checks for GPS against origin waypoints and internal navigation systems. (Successful initiation of checks; successful completion of checks; knowledge of factors which would prompt inflight position initialization and rechecks).

When designing a simulation to test the competency of an IA system, in addition to the level of training and “experience” the IA has acquired, the software must simulate a multitude of varying individual and contextual factors across the same operational test scenario. At a minimum, the simulation should define and proffer variants of the following variables:

- What is the starting system state? (Fuel levels, system status, weight and balance, aircraft capabilities)
- What is the operational and physical environment (or perception of the environment)? (Pilot health/status, communications, weather, airspace congestion, temporal constraints)
- What are the goals of the pilot/IA?
- Which properties of the system are most salient at this time?
- How many/what other responses or actions are available to the pilot/IA?
- How experienced or well-trained is the pilot/IA on this system?
- What are the pilot’s/IA’s beliefs and expectations of the aircraft systems (beyond rule-based training and system manual)?
- What level of trust does the pilot/IA have in the system?
- What level of SA does the pilot/IA have of the system’s state?
- What is the SOP/FAA/pilot manual guidance for this situation? How clear and/or stringent are the rules?
- What are the pilot’s/IA’s beliefs and expectations about the crewmate?

Knowledge checks are excellent simulation methods for addressing individual pilot competency, but the most crucial aspects of human-IA interaction must also be addressed using crew modeling and simulation techniques.

7.5.2 Understanding the Human Crewmate

Developing simulation techniques for evaluating and improving the IA’s understanding of how humans work is a complicated endeavor. While considerable research has focused on enhancing human awareness of their machine partners, virtually none has investigated providing awareness of human teammate states to machine partners. For example, flight deck automation feedback and alerting schemes are static, assuming pilots are always highly vigilant and have complete knowledge of complex system mode logic including all current and future states. By contrast, human teammates tend to recognize teammate state and adapt communication patterns accordingly to improve likelihood that

information being shared is attended, understood and acted upon.

Mental models are constructs that humans build to make sense of their world and generate predictions about future states, based on the available information, sensory data and perception. Because of gaps in shared awareness, operator mental models become de-coupled from the system model and there are no means to detect the de-coupling or adjust. For example, on modern flight decks, pilots become unaware that automation is no longer providing some expected support (e.g. Asiana Flight 214). In a symbiotic team, the flight deck would infer from pilot behavior that the mental model de-coupling was occurring, identify the specific gap, communicate to correct the awareness gap, and, if needed for time-sensitive and safety-critical situation, act in the absence of an appropriate crew response.

Scheutz [117] developed a conceptual framework for developing a Shared Mental Model (SMM) for human-robot (or “artificial agent”) teams. SMMs in the context of human-human teams have more to do with establishing and maintaining common ground based upon mental models of the other team members’ future activities and actions. Team mental models are critical for understanding the dynamic changes of team goals and needs and for tracking the overall moment-to-moment state of the team. In human-IA teams, it is critical to design for the IA’s ability to create concepts and rules for those concepts in order to improve team coordination and performance. In developing simulations to evaluate human-IA communication and resource management, therefore, it is important to recognize, understand and evaluate the implementation of SMM computational frameworks which should be inherent in the software architecture of the IA.

Human and digital systems also have a persistent representation challenge—they don’t speak the same language. For example, in teaming with mobile semi-autonomous vehicles, humans spatially reason and communicate based on semantic/qualitative representation whereas their autonomous crewmates work based on metric/quantitative understanding of the world. Consider the overhead and severe limits of communicating with humans with no common spoken language; shared awareness is very difficult to achieve and maintain across this language gap. Complex, digital systems model and reason on the work domain in a manner that is fundamentally different than their human partners—thus requiring all interactions between agents to be translated.

IA simulations with Humans in the Loop (HITL) will further clarify the human response to IA intentions, assessments and actions. How can the human distract or prevent the IA from performing its actions? How will the IA respond to these distractions or unwanted control inputs? In what ways might the goals and approaches of the human differ from that of the IA system? In which cases might the human approach work best (i.e., when should the IA actor defer assigned tasks and responsibilities to the human actor)? How can we design for prompt and clear communication and negotiation between human and IA actors?

IA systems should be exposed to varied and thorough scenarios enough times for the system designers to reliably predict system behavior in any event. The system behavior (actions) should be clear and unambiguous to the human crewmate, and the IA should provide task identification and progress

feedback to the human actor. In cases wherein the IA's decision-making is too rapid or complex for the human to process, the IA intentions should be stated at a higher level than just the task level (e.g.: "I'm trying to stabilize the approach in unexpected wind shear." This information should be communicated promptly and clearly to the human actor such that he or she has a chance to respond or query the IA further, as time allows.

7.5.3 Crew Resource Management

Crew resource management (CRM), or the development and preservation of interpersonal communication, information management and shared decision making strategies in the cockpit, is critical to assuring safe and efficient operation, reducing error, avoiding stress and increasing efficiency in the cockpit. Widespread adoption of CRM practices by aerospace operators, including explicit processes to maintain a common understanding by both pilots, exemplifies this. Building a high-functioning human-IA team requires the same shared awareness to effectively collaborate and coordinate joint performance; however, intelligent systems are often opaque to humans, there is no awareness of human states by the IA, and/or there is no established common language. In a study of CRM-related ASRS reports (for dual-pilot operations), Bearman [22] characterized a categorization of communication disconnects leading to breakdowns in coordinated decision making:

- Operational disconnects occurred when there was either a difference between the actions of one party and actions expected by the other party or a mismatch in the plans that each party had about the physical operation of the aircraft.
- Informational disconnects occurred when there was a difference in the information that each party possessed.
- Evaluative disconnects occurred when there was a difference in the evaluation or appraisal of information that was available to both parties.

One can surmise that informational or evaluative disconnects often lead to operational disconnects, contributing to the overall breakdown. CRM errors between crewmates or between the crew and ATC were wholly or in part responsible for the majority of the reports assessed. The researchers also found that evaluative disconnects were significantly harder to resolve than the other types, and therefore the design of cooperative human-IA systems must induce a shared understanding of information. Simulation and system modeling should focus on understanding differences in evaluative methodology between the human and the IA, and suggest how to train common methods of "sizing up" the situation, given the same information and contextual cues. Finally, simulations should assess how human-IA teams build and maintain shared mental models.

7.5.4 Roles, Responsibilities and Task Management

Crew task management (CTM), including proper prioritization and allocation of tasks between crewmembers, is another critical factor in flight safety. In a review of NTSB and ASRS reports [31], task management errors occurred in 23% of the accidents (79/324) and 49% of the incidents (231/470) reviewed. Proper task management strategies address not only who is or should be performing a task, but that crewmember's progress on task, how many tasks are in his queue, the complexity of the task (and whether it is possible for the human or IA to "multitask"). These factors will determine the need for task switching or reallocation at any given time.

Task management is especially critical to human-IA interaction, because there is a wider disconnect between the way information is ingested, evaluated and used to support decision-making between a human and a machine. To understand this, it is helpful to look at several ASRS case studies of aircraft incidents which involved task management errors.

Case Study 1

In the first case, a DHC-8-400 (Q400) crew distracted by weather, turbulence, EFB usage and passenger safety, forgot to reset a low altimeter (29.11 to 29.92) while climbing to FL230 and were subsequently advised by ATC of an altitude deviation.

The aircraft had been delayed for maintenance before departure and had the passengers on board for an hour, so there was a temporal demand factor. There were multiple lines of thunderstorms and high winds present on departure, distracting the PF. The PM was having some problems with a frequency change (the reception was poor and they were given a wrong frequency) which took some time to resolve. Both pilots were distracted and failed to reset either altimeter, which resulted in a large indication error. As they checked on to the new frequency, the Controller asked what their altitude was, which is when they spotted and corrected the problem. The main factor in this event was attention tunneling; the PF was focusing on the EFB to find clear weather and the PM focused on frequency change. Four task management errors observed in this case included (1) low prioritization of the instrument monitoring task (failure to follow SOP on departure) which should have been a priority ("aviate") task, (2) excessive monitoring of the weather (PF) and radio (PM), (3) low resource allocation to the aviate and monitoring tasks, and (4) high workload. Contributing factors included the temporal demands imposed by the ground delay, severe weather, and stress (expressed by the PF in his report).

Case Study 2

In this case, B737-700 flight crew experienced difficulties attempting to practice a RNAV approach into JAX in Visual Meteorological Conditions (VMC). Control Wheel Steering (CWS) mode was inadvertently selected and the aircraft descended prematurely triggering a low altitude alert from

ATC and an EGPWS warning. A visual approach ensued.

It was the Captain's (PF) first attempt at practicing an RNAV approach and the first officer (PM) had only flown one before. The crew was on approach into JAX, and requested the RNAV (GPS) 31. ATC cleared them direct to NIBLE (IAF); however they were past NIBLE when starting the turn which resulted in a larger turn than expected (virtually entering a downwind). When cleared for the RNAV approach, due to the close proximity to NIBLE, the autopilot was turning right towards the fix and shortly thereafter started a left turn towards POTME (close timing was a factor). At some point, the PF unknowingly disengaged LNAV and went into CWS. PM alerted him to the mode change, but he did not respond. The crew became disoriented; they were off course, confused with what the airplane was doing, and descending. The PF disengaged the autopilot and continued descending while flying away from the field. PM instructed him several times to stop descending, climb, and return to the glidepath. The PF, fixated with the new RNAV procedures, was unresponsive to the PM's verbal alerts. The AC had descended below the charted altitude at POTME (2,600 ft, 9 miles from TDZ) to about 1,700 ft MSL. ATC alerted the crew, who responded with "correcting," requested and were subsequently cleared for the visual approach. The main factors in this event were poor CRM, resource mismanagement and unfamiliarity with the RNAV procedures. The PM noted in his report that they should have prioritized flying the airplane above all else, and should have discontinued the RNAV approach earlier when they became disoriented. The PM felt he was assertive but nonetheless, the PF failed to heed his guidance. His fixation on the procedure and mode confusion caused him to ignore (or perhaps fail to hear) the warnings of his crewmate.

Four task management errors we observed; these include (1) high prioritization of the RNAV procedure, (2) excessive task monitoring (procedural as well as confusion about the AC control response), (3) failure to initiate the climb and redirect to course, and (4) failure to terminate the RNAV procedure and request the familiar VFR approach as soon as they became disoriented. Other contributing factors included inadequate training, poor CRM and spatial disorientation.

Case Study 3

In the third example, an A320 Captain on the LAS TYSSN THREE RNAV calculated his descent constraint compliance incorrectly and failed to meet one of the constraints before realizing his error, even though his First Officer was cautioning about the error.

Enroute to LAS, ATC cleared the crew to descend to FL260 at 280 KIAs. The crew leveled off at FL260 and set cruise mode for a managed descent speed of 282 KIAs; however, the PF erroneously selected 280 on the FCU instead of 280 KIAs. ATC cleared them to descend via the RNAV arrival for a visual approach to Runway 25L. The PF dialed in 8,000 to meet the altitude restriction at PRINO. Both crewmembers had triple

checked all the altitude constraints on the arrival, and both had the constraints button pushed. The first waypoint had an “at or above FL200” constraint and the subsequent waypoint had an “at or below FL190” constraint. They engaged managed descent and calculated that they needed 21 miles to get below FL190 over the second waypoint (19 nm away from the first). The aircraft started a 900 FPM descent, still in selected speed of 280. The MCDU showed approximately 2,400 FT low on the path but slowly decreasing deviation. **The PM noticed and informed the PF about the navigation errors, but due to attention tunneling (focusing on the altitude and descent rate calculations), he failed to adhere to proper procedure to correct the descent.**

Three main task management errors were observed, including (1) low prioritization of the aviate task, (2) late initiation of the corrective measure, and (3) low/poor resource allocation in not prescribing the troubleshooting to the PM. Contributing factors included distraction, over thinking on the part of the PF, and poor CRM overall.

The three cases presented above illustrate the impact of task management errors on the pilots’ ability to properly navigate, adhere to published procedures and maintain stability on the flight path. Procedural compliance and FPM are directly impacted by crew task management errors.

Another important consideration in the assessment of a cooperative human-IA crew is that of role-setting. The relationship between a crew of two human pilots in terms of responsibilities and roles may vary from day-to-day and even during a single flight. There are four cornerstone concepts relating to the design of cooperative man-machine systems [52]. These are: ability, authority, control and responsibility (Figure 23). Consistency in the relations between these concepts has been identified as an important quality for the system design.

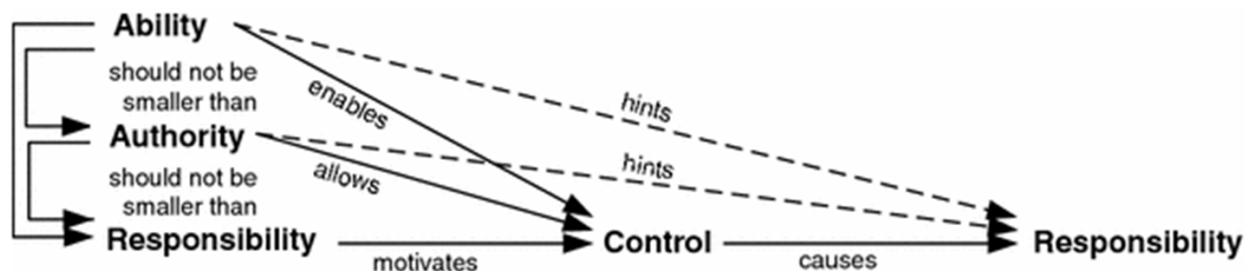


Figure 23: Authority and Autonomy in Cooperative Human-IA interaction

But IA systems designed to work interchangeably with the human crewmate must be more adaptable to changing situations. Simulations should test to ensure that the requirements expressed in the figure above are met for both the human pilot and the IA system, across all operational scenarios. To this end, simulations designed to validate the IA must not only ensure that the highest level of responsibility can be assumed by the IA, but that the highest level of responsibility may also be assumed out by the human pilot, and the IA system will react in an appropriately dominant or passive manner.

7.5.5 Fault Management

Another human factors issue which must be addressed by the safety assessment is fault management. Fault Management is a process utilized by both humans and systems. The term “Fault Management” can refer to the engineering of a system to prevent, detect, isolate, diagnose, and respond to the detection of faults within a system or subsystem [100]. Fortunately, as aircraft systems have evolved, accidents related to true system “faults” have substantially decreased. Automation has taken a more active and authoritative role in system monitoring, diagnosis and recovery, such that the more serious consequences occur when the human-automation relationship breaks down. This can happen in a variety of ways. Cognitive mismatches are breakdowns in the pilot’s mental model of the situation, aircraft orientation, or system/automation state. Errors may occur as a result of poor design and/or training with respect to fault monitoring, detection and alerting systems. Pilot errors are also often caused by inappropriate or poorly defined function allocation. The allocation of responsibilities between the pilot and the automation in a given situation must appropriately consider the pilot’s workload, situational awareness, information requirements and reaction time.

Mental models help pilots to determine which system and state variables to monitor. The values of these variables aggregate to establish their mental models of the aircraft state, flight controls and subsystems. However, the crew’s mental model, or perceived state, of what is happening does not always match reality. This is an example of a cognitive mismatch. Cognitive mismatches can fall into two categories. A real mismatch occurs when the pilot’s mental model does not match reality; a perceived mismatch occurs when the pilot perceives the system is not operating as expected, when in reality their mental model is correct (i.e., a “false alarm”). If detected, a mismatch can often be resolved through some diagnostic activity. Left undetected, the situation may deteriorate.

Case study: Baxter, Besnard & Riley [21] evaluated an accident at Nagoya, Japan, in which the aircraft landed tail-first on the runway. In this case, there was an accidental engagement of the “Go-Around” mode just before landing. In this mode, the aircraft applies maximum thrust and climb rate to regain altitude. When the Go-Around mode caused the aircraft to climb above the glide path, the First Officer (FO), acting as PF, tried to bring the nose down and decrease thrust but due to its settings the aircraft resisted control inputs. The PF then tried engaging the autopilot and as it pitched up again, he disengaged. The Captain took over the controls shortly before impact and expressed concern about the aircraft’s lack of response. Finally the aircraft stalled and the crew was unable to recover.

The cognitive mismatch is clear from the FO’s attempts to counteract the Go-Around mode instead of disengaging it. In this case, the situation was worsened by the flight deck automation which did not allow yoke force to disengage the autopilot below 1500 feet.

Cognitive mismatches can occur as a result of aircraft automation configuration, which can vary greatly between aircraft manufacturers. At a high level, Boeing aircraft systems require the pilots to

decide upon the action, Airbus pilots are advised what action to take, and McDonnell Douglas pilots are informed after the action has been taken.

Sheridan and Parasuraman [122] describe a simple analysis, based on expected value decision theory, for deciding whether a human or automation is best for a failure detection task. A simple failure-response task which derived from a set of binary circumstances (failure or normal) and was subject to binary responses (action, no action) was evaluated in terms of all four possible outcomes (true positive, true negative, false negative, and false positive) using an expected-value analysis. This calculation determines when to automate based on which actor (human vs. automation) will provide the greatest expected value (benefit-cost). In the basic equation, the outcomes enumerated above combine to constitute the expected value. Each of the four outcomes is composed of three factors: a priori probability of actual state, conditional probability of correct or incorrect decision/response, and benefit or cost of correct or incorrect decision/response. Variations of this equation are derived based on variations in the weighting of each possible outcome (e.g., minimizing false positives), and the appropriate analysis would be largely dependent on the nature of the task and the delineation of what constitutes a “benefit”. The authors’ recommendation therefore is to “simplify one’s problem as much as possible, estimating benefits and costs at least in proportion to one another” to make the decision analysis most effective. The authors also discuss other criterion such as the minimax-loss criterion, in which an actor is chosen based on its ability to minimize the worst case payoff. The analysis described provides a rational, quantitative basis for the decision of whether or not to automate. This analysis can applied to dynamic task allocation tools, and can also be extrapolated to non-binary automation paradigms where there are multiple LOA by simply applying the calculation to each individual level.

7.5.6 Impact of Human Automation Teaming (HAT) on V&V Methods

Traditional systems engineering requirements address the V&V of integrated systems and their hardware and software components. However, they do not always address the influence of a dynamic, autonomous machine partner on human performance. The pure systems-centered approach does not accurately account for the pilot’s role in the overall safety of the system. Further, certification requirements such as “flight deck systems shall enable flight crew awareness” (14 CFR 25.1302) do not adequately address the challenges related to airplane state awareness. While the proposed new system requirements could technically be covered by existing, generic requirements, the level of detail is not sufficient for use in design and evaluation [139]. Such generic requirements should be augmented with more detailed system requirements based on the wealth of available human factors research into human-IA interaction and shared awareness.

Current V&V practice dictates tests run until failure, but humans can “fail” and then subsequently recover. The dynamic interaction between pilot collaborative with intelligent system, mental model, and HAT user interfaces requires a new methodology to handle the explosion in the problem space. Fortunately, the FAA recognizes this bias towards system-centric process and has made strides to

release human factors regulations intended to promote a more human-centered design approach.

Human factors requirements in general also create traceability challenges for the hardware and software designers. The high-level requirements defined in the various regulatory documents discussed above are decomposed into functional requirements and applied to hardware components and software specifications in a manner that can be traced from each element back to the high level requirement. However, human factors requirements, especially those addressing psychological constructs, can only be realistically/meaningfully tested at an integrated level, a challenge for programs developing advanced collaborative systems. This poses challenges for the de-constructive V&V process at the lowest levels of decomposition. This is especially the case for constructs such as situational awareness that is critical for HAT, which emerges from the perception, comprehension, and projection of elements within the environment. At some point in the decomposition process, psychological constructs are no longer verifiable creating traceability issues for requirements managers and V&V engineers.

8 Future Work Directions for Safety Assurance of IA Systems

In the previous sections, we have presented the challenges posed by IA systems and the various aspects of assuring the safety of such system: including assurance methodology/processes, system requirements impacting safety, dependability and system architecture considerations, and V&V capabilities needed. We have also explored changes needed across these aspects of safety assurance and potential approaches to mitigate the challenges and associated safety concerns. Section 3.1 and Figure 3 provide a notional structure of the how the different aspects of safety and V&V related processes will be integrated to form a basis for an assurance case. In this section, we describe areas of future research that needs to be conducted collaboratively in the stakeholder community of regulators, researchers, tool-providers, and the industrial makers of IA systems.

8.1 Top-Down Systems Approach for Safety Assurance of IA systems

As we have observed in Section 2.4, the current assurance approach prescribes fixed V&V process activities, criteria, and metrics via regulatory guidance provided in ARP4754A [127] and DO-178C [104]. This has worked well for the assurance of traditional avionics and control systems. However, given the increasing complexities of modern distributed systems and the new dynamic, “intelligent” functions of next generation of IA systems, we recommend the following areas of investigation as part of a “top-down” systems approach:

- **Verification techniques and coverage metrics**

Full requirements driven coverage of generic neural network implementations may be insufficient to mitigate complexities of the behavior which emerges as the code is configured with run-time weights. Similarly, requirements driven assurance of table data values, may not be applicable to the neural network data tables, given that the gap between a value and its contribution to the emergent system behavior is very large, and potentially intractable to address. To mitigate these issues, new metrics to gauge the effectiveness and completeness of training sets need to be developed. This may be a very fruitful area for future work. In the ideal, new metrics to support some systematic assessment of the learned capabilities can be developed. It is important to stress, that such metrics need to be developed from normal and abnormal conditions. Characterization of the anticipated system component failures and potential life-time degradation of their function should also be systematically measured and assessed, along with the anticipated environmental conditions. Measures to assess the fitness of the training set in each of these aspects would be beneficial. However, a careful balance needs to be maintained to ensure that the verification activities remain cost effective.

Given some recent developments such as the machine-learning-based auto-pilot of Baomar et al. [17], another thread of potential research would be to utilize traditional assurance approaches in conjunction with the machine learning component. For example, as part of traditional autopilot

lot design validation an extensive suite of Monte Carlo based simulation campaigns are used to evaluate the system performance. Accessing the behavior of a machine-learned autopilot using this environment may be educational in terms of what type of coverage metrics make sense.

- **Using extension of STPA technique as part of a comprehensive safety analysis**

In addition to revision of component level assurance guidance, we have also stressed in this report that a systems context is required to better address the machine-learned component interactions. Using frameworks such as STPA to better capture and examine a component's control actions in relation to the larger system level safety contexts may be very beneficial. Of particular interest is to capture how the influence of a machine-learned component is captured and reasoned within the hierarchical control structure. How, unsafe control actions are mitigating using architectural and/or human/procedural support mechanisms would also benefit from systematically driven scrutiny. In this regard, we believe that leveraging the recent STPA extensions to address multiple component interactions [132] and human to computer interactions [54] will be greatly beneficial, should representative case studies be developed and demonstrated. Furthermore, integrating the STPA analysis in conjunction with the pervasive monitoring strategies (discussed later) may also be of great benefit, improving the traceability between the run-time assurance and the design time control action assumptions and constraints. Hence, establishing some systematic methods to integrate and link the safety assessment frameworks is needed. An important aspect of such an integration is the interplay between the hosted platform hardware architectures, especially with respect to the failure assumptions of the hardware components.

- **Constructing a multifaceted assurance case for IA system safety**

As we noted in Section 3, the assurance case will need to be multi-faceted to address the different types of arguments/processes, variety of subsystem implementation choices (e.g., rule based, ML), V&V techniques, and architectural provisions needed to assure safety. Furthermore, the assurance case composition should address the following concerns/challenges brought by the different aspects of IA functionality:

- Some form of requirements/constraints will still be needed for machine learning system components in addition to “safety invariants.” They would be specific to the ML technique and not follow prescriptive guidance. Such requirements should be established as part of the assurance case (e.g., based upon flight rules).
- Hierarchical functional composition of the system would enable applying “prescriptive guidance” at higher levels where traditional logic and rule-based processing is employed, but an argument-based approach at the lower levels where ML techniques are used.
- A combination of prescriptive and argument-based approach is also applicable to human-IA interactions that are described later in this section.
- The mitigation of certain safety hazards by pervasive monitoring must be part of the assurance case.

8.2 Pervasive Monitoring as a Major Component of overall System Assurance

A large component of architectural assurance will be pervasive monitoring. A pervasive monitor is an independent system which may utilize the same sensors as the IA, but have different interpreters. The monitor checks the actual behavior of the controlled plant (self-driving car, IA plane etc.) against top-level (hence “pervasive”) constraints. Section 6.3 provides an overview of the pervasive monitoring concept and challenges.

The monitor needs to know what the bounds of acceptable behavior are for the current situation. Obviously, the monitor must be able to determine all of the important features of the current situation and do it independently of the IA. Once the monitor obtains this required situation information, it must determine if the current IA behavior is acceptable and intervene if it isn't. This intervention would compute “reversionary” commands, restricting behavior and plant performance to a much more constrained envelope.

Such a monitor would need to be dependable as discussed in section 6.3. Designing a monitor with the complexity needed to cope with all critical situations and still be simple enough to verify is a daunting challenge. It would be advisable to start out with baby steps, i.e. build a small pervasive monitor with an initial set of rules that are achievable in the near term. The rules could be derived from legal or policy documents (e.g., the California vehicle code or FAA “rules of the air”).¹⁴ Designing this simple monitor includes how to code these rules into it, which would be a research project in its own right. Answer-set programming is one approach that looks promising [28].cardiac care example).

8.3 Safety Assurance Approach for Human-IA Interaction

There are several recommended areas for continued research as well as the development of human interface requirements and conceptual design. Human-automation interaction research needs to evolve in parallel with the evolution of IA. Of importance to the development and the safety verification for intelligent systems are the issues of trust in automation, crew resource management, communication styles, natural language interaction and shared situation awareness (SA) and shared mental model (SMM). Potential research topics are outlined below:

- **Adapting explainable AI (XAI) concepts to flight deck interactions**

Future research should focus on developing XAI strategies and subsystems to support the pilot's understanding of the “thought” processes of the IA system. The Theory of Mind (ToM) allows humans to attribute mental states—beliefs, intents, desires, goals and knowledge—to other humans. The importance of XAI and CRM in human-automation interaction is not only to improve trust between human and IA entities, but also to foster a ToM by which the human

¹⁴While there may be significant differences in de facto versus de jure behavior for car drivers, this may be less true for the highly regulated airspace.

interacts with the IA as an individual with perspectives of its own. By studying ToM and applying this philosophy to the design of XAI interfaces, we can develop a more natural state of understanding, trust and teamwork.

- **SMM and Shared SA**

A great deal of research has centered on situation awareness and mental models across human-automation teaming and mode awareness, particularly in the aviation arena. Prior research has focused on the human pilot remaining cognitively involved in aircraft operations, even when the automation is doing the work. The human pilot would be continuously monitoring the automated systems. As increasingly intelligent systems place the human pilot less and less at the forefront of aircraft operations and decision-making, we need to reevaluate our approaches to understanding and fostering these concepts. We need to start assuming the human pilot is not cognitively engaged, and therefore will require a period of “getting up to speed” with not only the situation but also the comprehension level of the IA and the stage at which the system is making decisions. HITL testing, cognitive walkthroughs and post-simulation interviews can help us better understand where shared awareness breaks down between the human and the IA.

- **Trust in Automation and Crew Task Management**

There are multiple potential areas of study here. As discussed earlier, XAI can go a long way to establishing a level of trust between the human and the IA, because it allows the pilot to understand the motivation behind the IA’s decisions. Developing a ToM for IA will also stimulate trust and improve task management. In addition to XAI and ToM, an emerging research focus in Human-Automation Teaming (HAT) has designers working to develop collaboration strategies for human-robot teams by utilizing ML and probabilistic methods.

- **Communication styles and Natural Language Processing (NLP)**

A crucial design issue for collaborative automation is displaying and projecting information and plans of action to the human pilot in terms which are understandable and to which the human can naturally respond. To accomplish this, the system should endeavor to “speak” to the human pilot in a way that another human would convey this information, and to allow the human teammate to articulate his or her thoughts without the added workload of conforming to a strict communication structure or vernacular. System designs should support airline SOP for crew communications, ATC communications, as well as shortened responses, conversational tones and colloquialisms.

- **Crew Resource Management (CRM)**

It is important to research CRM in the context of IA systems, identifying which strategies will work best as-is and which will require an evolutionary process to support IA crew interaction. Some of these solutions will only become apparent as the human element is exposed to a full range of operational scenarios interacting with a state-of-the-art IA partner. Intermediate solutions can be developed, however, based on HITL testing as while accomplishing the same tasks

using different crew interaction strategies. Strategies for workload and task management (loading, shedding and interruption) can also be developed based on Human-in-the-Loop (HITL) experimentation and ML-based predictive technology to determine the best cooperative relationship.

- **Individual Differences**

Further research is needed in adapting IA to work seamlessly with a range of intellect, emotion, confidence, spontaneity and rationality in its human partners, and to account for varying levels of stress, anxiety and fatigue from day-to-day and moment-to-moment. Due to the large range of individual differences in human pilots, it will be important for the technology to develop to identify personality traits as well as physical, cognitive and emotional states of the human and respond accordingly. To do this, we can employ such methods as individual pilot identification software and psychophysiological monitoring systems. Studying the stress response can help link human psychological states to situational rationales.

8.4 Automated Reasoning to Assist in Architecture and Safety Analyses

In today's world, the pace of technology innovation is fueling a relentless increase in system complexity. New applications, increasing connectivity, and new levels of autonomy are enabling new application areas such as fully and partially autonomous cockpits that were unimaginable a few years ago. In prior funded research, we concluded that the ability of a human alone to effectively reason about the safety of modern complex systems is at a tipping point. Although new methods, such as Systems-Theoretic Process Analysis (STPA), can be argued to improve functional and system hazard identification, more effective computer support and model integration is needed to mitigate the inherent limitations of human reasoning. Furthermore, in the modern globalized world, the complexities of how we build, in addition to what we build, are growing increasingly important. The developmental guidance, such as DO-178C, has also grown in size and complexity from one document to multiple dedicated supplements—as the new risks of modern-development paradigms such as model-based, object orientated, formal method and tool assisted work-flows are mitigated. Therefore, the integrated reasoning of both how and what we build is needed to completely discharge the full intent of system safety and design assurance. Such an integrated approach is beyond the reach of traditional formal methods and model-based system engineering technologies. However, we believe that recent developments within the realm of logic model processing, co-inductive, deductive, and abductive logical reasoning will yield a paradigm shift and a step-change in the quality of system analysis and system safety assurance. Just as the IBM Watson technology is revolutionizing multiple industries, we believe that logically assisted reasoning technologies may radically transform safety-critical system development and design assurance.

The logic-based process modeling system should integrate co-inductive, deductive, abductive, and constraint reasoning for naturally modeling and certifying assurance of complex behaviors that mod-

ern avionics systems exhibit. The ability to naturally and elegantly model systems is the key factor that greatly facilitates certifying safety and assurance claims. Co-inductive reasoning is needed to model cyclical and rational infinite behavior that arises in various system modeling formalisms (for example, controllers that run forever). Abductive reasoning is needed to answer what-if questions and counterfactual conditional queries (of the type “if this did not occur, then this would not have happened”) that system designers are interested in. Constrained reasoning is needed to faithfully model the real-time aspects of systems. In addition, co-induction is critical for modeling negation as failure and default reasoning, both of which are needed for representing the knowledge in the natural language interfaces discussed above.

From the integrated framework, computer-assisted analyses and queries could support the key system liveness and safety properties to be validated. Human-guided semi-automated assistance frameworks, such as STPA, Control Action causality analyses, FTA, and FMEA will also be available to improve and de-risk many of the manual activities performed today. Leveraging industry standard model notations and constrained natural language based hazard and requirement capture, the formal ‘backend’ of the proposed framework will be largely transparent, thus lowering the barrier of entry and pragmatic industry acceptance. Integrated reasoning of system architecture and system behavior (including failure modes)—within a development-process-aware context that links functional and item design assurance levels (FDAL/IDAL) to the item and function aware process objectives—will support unprecedented system assurance analysis and guided examination.

8.5 Autonomous Ethics

Due to the complexity of the subject matter, the issues of ethics and morality for autonomous vehicles haven’t made much headway, particularly with respect to any kind of implementation. It has only been the last few years that these issues have been discussed, from a philosophy viewpoint. These discussions range from simplistic variations of utilitarianism views on the “Trolley Car” dilemmas (e.g., killing a Nobel prize winner versus a dozen drug dealers) to discussions of utilitarianism versus deontology and the moral imperative for self-preservation. Some thorny obscure issues can arise, e.g., should an autonomous system ever become a vigilante? Once the difficult task of establishing autonomous ethics is done, the next difficult task is implementing the enforcement of ethics. Autonomous ethics enforcement cannot be implemented in standard expert systems, there just too many scenario variables. Autonomous ethics also cannot be developed via machine learning. How many people would have to be killed for a machine to determine that it shouldn’t take a particular action? In addition, even if this learning were possible, what would be learned would be morality, not ethics. Ethics needs to be externally taught. How would this be done? Finally, we get what may be the most difficult task of all: How can we assure that an autonomous ethics enforcement design does what it needs to do?

9 Conclusions

We have presented here the safety assurance considerations for IA systems, including the shortfalls of current methods and the new challenges posed by IA systems when human and automation roles and flight-safety responsibilities are shared. This impacts several areas of safety assurance: including safety assurance methods/processes, safety-related requirements of IA systems, system architecture considerations, and V&V techniques.

Given the projected complexity increase associated with the next generation of IA system, it is expected that new methods and technologies will be required in all of the aforementioned areas. We have examined specific issues and potential approaches in these areas to address the challenges posed by IA systems. Several important consideration emerge from this work—some of potential future research directions are described in Section 8.

Finally, we believe that a broader discussion of such issues is needed in the stakeholder community—leading to relevant, focused research work in this area. This will also support the development of acceptable standards for such systems. Due to the new and novel functions and features expected as IA applications are introduced, certification experts should acquire additional skill sets and familiarization with automation technology and functions. Such collaboration across research community, industry, and regulatory bodies is needed to support the design, development and certification of IA systems.

References

- [1] Adelard LLP, London, UK. *ASCAD: Adelard Safety Case Development Manual*, 1998. Available from <https://www.adelard.com/resources/ascad.html>.
- [2] Federal Aviation Administration. Electronic code of federal regulations - general operating and flight rules. *Electronic Code of Federal Regulations*, 2017.
- [3] Seth Ahrenbach. Formal analysis of pilot error using agent safety logic. Master's thesis, University of Missouri, May 2016.
- [4] Seth Ahrenbach. Reasoning about safety-critical information flow between pilot and computer. In *NASA Formal Methods*, volume 10227 of *Lecture Notes in Computer Science*, pages 342–356, Mountain View, CA, May 2017. Springer-Verlag.
- [5] Seth Ahrenbach and Alwyn Goodloe. Formal analysis of pilot error with agent safety logic. *Innovations in Systems and Software Engineering*, 14(1):47–58, March 2018.
- [6] Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. *arXiv preprint arXiv:1708.08611*, 2017.
- [7] Erin Alves, Devesh Bhatt, Brendan Hall, Anitha Murugesan, and John Rushby. Considerations for safety assurance methods for increasingly autonomous systems. Technical report, Deliverable 2: Technical Report for NASA Project Assurance Reasoning for Increasingly Autonomous Systems (ARIAS), 2017.
- [8] Erin Alves, Devesh Bhatt, Brendan Hall, Anitha Murugesan, and John Rushby. Requirements and assurance considerations for increasingly autonomous systems in reduced crew operations. Technical report, Deliverable 1: Technical Report for NASA Project Assurance Reasoning for Increasingly Autonomous Systems (ARIAS), 2017.
- [9] Paul Eric Ammann and John C Knight. Data diversity: An approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, 1988.
- [10] D. Amodei et al. Deep speech 2 end to end speech recognition in English and Mandarin. In *International Conference on Machine Learning (ICML)*, pages 173–182, 2016.
- [11] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.
- [12] T. Anderson and R. W. Witty. Safe programming. *BIT*, 18:1–8, 1978.
- [13] Anish Arora and Sandeep S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In *18th International Conference on Distributed Computing Systems*, pages 436–443, Amsterdam, The Netherlands, 1998. IEEE Computer Society.
- [14] SAE ARP. 5107 (aerospace recommended practice): Guidelines for time-limited-dispatch analysis for electronic engine control systems. Rev. B. *Society of Automotive Engineers*, 2006.

- [15] Algirdas Avižienis. The *N*-Version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, December 1985.
- [16] Haitham Baomar and Peter J. Bentley. An intelligent autopilot system that learns flight emergency procedures by imitating human pilots. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–9, Athens, Greece, December 2016.
- [17] Haitham Baomar and Peter J. Bentley. An intelligent autopilot system that learns piloting skills from human pilots by imitation. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1023–1031, Washington, DC, June 2016. IEEE.
- [18] Haitham Baomar and Peter J. Bentley. Autonomous landing and go-around of airliners under severe weather conditions using artificial neural networks. In *Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 162–167, Linköping, Sweden, October 2017. IEEE.
- [19] Haitham Baomar and Peter J. Bentley. Autonomous navigation and landing of airliners using artificial neural networks and learning by imitation. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, Honolulu, HI, November 2017.
- [20] Samik Basu and CR Ramakrishnan. Compositional analysis for verification of parameterized systems. *Theoretical Computer Science*, 354(2):211–229, 2006.
- [21] Gordon Baxter, Denis Besnard, and Dominic Riley. Cognitive mismatches in the cockpit: Will they ever be a thing of the past? *Applied ergonomics*, 38(4):417–423, 2007.
- [22] Christopher Bearman, Susannah BF Paletz, Judith Orasanu, and Matthew JW Thomas. The breakdown of coordinated decision making in distributed systems. *Human factors*, 52(2):173–188, 2010.
- [23] Sagar Behere and Martin Törngren. A functional architecture for autonomous driving. In *Proceedings of the First International Workshop on Automotive Software Architecture*, pages 3–10. ACM, 2015.
- [24] Christian Berger and Bernhard Rumpe. Engineering autonomous driving software. *arXiv preprint arXiv:1409.6579*, 2014.
- [25] J. Anthony Blair. What is informal logic? In Frans H. van Eemeren and Bart Garssen, editors, *Reflections on Theoretical Issues in Argumentation Theory*, volume 28 of *The Argumentation Library*, pages 27–42. Springer, 2015.
- [26] Ronen I Brafman and Moshe Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.
- [27] Bureau d’Enquêtes et d’Analyses (BEA), Paris, France. *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*, July 2012.

- [28] Zhuo Chen, Elmer Salazar, Kyle Marple, Gopal Gupta, Lakshman Tamil, Sandeep Das, and Alpesh Amin. Improving adherence to heart failure management guidelines via abductive reasoning. 07 2017.
- [29] Chih-Hong Cheng et al. Neural networks for safety-critical applications—challenges, experiments and perspectives. *arXiv preprint arxiv:1709.00911*, September 2017.
- [30] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. *arXiv preprint arxiv:1705.01040*, April 2017.
- [31] Chung-Chi Chou, Das Madhavan, and Ken Funk. Studies of cockpit task management errors. *The International Journal of Aviation Psychology*, 6(4):307–320, 1996.
- [32] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning assumptions for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.
- [33] RV Conference. Runtime verification home page. <http://www.runtime-verification.org/>, 2018.
- [34] National Research Council et al. *Autonomy research for civil aviation: toward a new era of flight*. National Academies Press, 2014.
- [35] Kenneth Craik. *The Nature of Explanation*. Cambridge University Press, Cambridge, UK, 1943.
- [36] J. Croft. Nasa advances single-pilot operations concepts. *Aviation Week & Space Technology*, Jan 2015. Available at <http://aviationweek.com/technology/nasa-advances-single-pilot-operations-concepts>.
- [37] Jacob L Cybulski. The formal and the informal in requirements engineering. Technical report, Technical Report 96/7, Department of Information Systems, The University of Melbourne, 1996.
- [38] Hal Daume III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, 2006.
- [39] Peter DeSalvo and Daniel Fogarty. Safety issues and shortcomings with requirements definition, validation, and verification processes final report. Technical Report DOT/FAA/TC-16/39, Federal Aviation Administration, December 2016.
- [40] RTCA DO. 254. *Design Assurance Guidance for Airborne Electronic Hardware*, 2000.
- [41] Dov Dori et al. *Model-based systems engineering with OPM and SysML*. Springer, 2016.
- [42] Kevin Driscoll. Real system failures. NASA Dashlink, November 2012. <https://c3.nasa.gov/dashlink/resources/624/>.

- [43] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.
- [44] Zachary T. Dydek, Anuradha M. Annaswamy, and Eugene Lavretsky. Adaptive control and the NASA X-15-3 flight revisited. *IEEE Control Systems Magazine*, 30(3):32–48, March 2010.
- [45] Bureau d’Enquêtes et d’Analyses. 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. Paris: BEA, 2012.
- [46] *Aviation Handbooks & Manuals*. https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/.
- [47] Boeing FAA. Safety Issues with Requirements Definition, Validation, and Verification Processes. 2014.
- [48] José M. Faria. Machine learning safety: An overview. In *Proceedings of the 26th Safety-Critical Systems Symposium*, York, UK, February 2018.
- [49] Federal Aviation Administration. *System Design and Analysis*, June 21, 1988. Advisory Circular 25.1309-1A.
- [50] Lu Feng, Marta Kwiatkowska, and David Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *International Conference on Fundamental Approaches to Software Engineering*, pages 2–17. Springer, 2011.
- [51] Michael Fisher, Louise Dennis, and Matt Webster. Verifying autonomous systems. *Communications of the ACM*, 56(9):84–93, September 2013.
- [52] Frank Flemisch, Matthias Heesen, Tobias Hesse, Johann Kelsch, Anna Schieben, and Johannes Beller. Towards a dynamic balance between humans and automation: authority, ability, responsibility and control in shared and cooperative control situations. *Cognition, Technology & Work*, 14(1):3–18, 2012.
- [53] Flight Safety Foundation. *A Practical Guide for Improving Flight Path Monitoring: Final Report of The Active Pilot Monitoring Working Group*, November 2014.
- [54] Megan Elizabeth France. *Engineering for humans: A new extension to STPA*. PhD thesis, Massachusetts Institute of Technology, 2017.
- [55] Deborah Frincke, Dave Wolber, Gene Fisher, and Gerald C. Cohen. Requirements specification language (rsl) and supporting tools. Technical report, NASA, 1992.
- [56] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods. *AAAI 2018*, 2018.
- [57] Javier Garcia and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

- [58] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1037–1044. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
- [59] Gabriel Gelman, Karen Feigh, and John Rushby. Example of a complementary use of model checking and human performance simulation. *IEEE Transactions on Human-Machine Systems*, 44(5):576–590, October 2014.
- [60] Shalini Ghosh, Patrick Lincoln, Ashish Tiwari, and Xiaojin Zhu. Trusted machine learning for probabilistic models. In *Reliable Machine Learning in the Wild at ICML*, New York, NY, June 2016. The 33rd International Conference on Machine Learning (ICML 2016).
- [61] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016.
- [62] Leo Groarke. Deductivism within pragma-dialectics. *Argumentation*, 13(1):1–16, 1999.
- [63] Leo Groarke. Informal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2017 edition, 2017.
- [64] DoD Architecture Framework Working Group et al. Dod architecture framework version 2.0, volume 1: Introduction, overview, and concepts. *Washington US: Department of Defense*, pages 1–90, 2009.
- [65] David Gunning. Explainable artificial intelligence (xai). <https://www.darpa.mil/program/explainable-artificial-intelligence/>, 2018.
- [66] Philipp Helle, Wladimir Schamai, and Carsten Strobel. Testing of autonomous systems—challenges and current state-of-the-art. In *INCOSE International Symposium*, pages 571–584. Wiley Online Library, 2016.
- [67] Jaakko Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press, Ithaca, NY, 1962.
- [68] C. Michael Holloway. Making the implicit explicit: Towards an assurance case for DO-178C. In *International System Safety Conference 2013*, 2013.
- [69] Gergory J. Holt. The certification challenge of the high technology aircraft of the 1990’s. In *Aerospace Technology Conference and Exposition*, SAE Technical Paper Series, Paper 871842, Long Beach, CA, October 1987. Society of Automotive Engineers.
- [70] Albert L. Hopkins, Jr., T. Basil Smith III, and Jaynarayan H. Lala. FTMP: A highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.
- [71] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

- [72] IEC ISO. IEEE: ISO/IEC/IEEE 42010: 2011-systems and software engineering–architecture description. *Proceedings of Technical Report*, 2011.
- [73] Vijay Manikandan Janakiraman. Explaining aviation safety incidents using deep learned precursors. *arXiv preprint arXiv:1710.04749*, 2017.
- [74] Vijay Manikandan Janakiraman, Bryan Matthews, and Nikunj Oza. Finding precursors to anomalous drop in airspeed during a flight’s takeoff. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1843–1852, Halifax NS, Canada, August 2017.
- [75] Vijay Manikandan Janakiraman, Bryan Matthews, and Nikunj Oza. Using ADOPT algorithm and operational data to discover precursors to aviation adverse events. In *AIAA Infotech@Aerospace Conference*, pages 1638–1651, Kissimmee, FL, January 2018.
- [76] Daniel P. Jenkins et al. *Cognitive work analysis: coping with complexity*. Ashgate Publishing, Ltd., 2009.
- [77] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang, and Myoungho Sunwoo. Development of autonomous car—part i: Distributed system architecture and development process. *IEEE Transactions on Industrial Electronics*, 61(12):7131–7140, 2014.
- [78] Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(4):596–619, 1983.
- [79] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *35th AIAA/IEEE Digital Avionics Systems Conference*, Sacramento, CA, September 2016. The Institute of Electrical and Electronics Engineers.
- [80] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.
- [81] Tim Kelly. *Arguing Safety—A Systematic Approach to Safety Case Management*. DPhil thesis, Department of Computer Science, University of York, UK, 1998.
- [82] David Krueger et al. Deep nets don’t learn via memorization. *ICLR Workshop track*, 2017.
- [83] Safety-Critical Systems Research Lab. SAHRA - STPA based hazard and risk, safety-critical systems research lab team of ZHAW Zurich University of Applied Sciences. <http://www.sahra.ch>, 2018.
- [84] Edith LM Law, Melanie Coggan, Doina Precup, and Bohdana Ratitch. Risk-directed exploration in reinforcement learning. *Planning and Learning in A Priori Unknown or Dynamic Domains*, page 97, 2005.
- [85] Nancy G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011. Google-Books-ID: 0gZ_7n5p8MQC.

- [86] Bev Littlewood and John Rushby. Reasoning about the reliability of diverse two-channel systems in which one channel is “possibly perfect”. *IEEE Transactions on Software Engineering*, 38(5):1178–1194, September/October 2012.
- [87] Bo Liu, Yanshan Xiao, Longbing Cao, Zhifeng Hao, and Feiqi Deng. Svdd-based outlier detection on uncertain data. *Knowledge and information systems*, 34(3):597–618, 2013.
- [88] D-RisQ Ltd. Modelworks. <https://www.digitalmarketplace.service.gov.uk/g-cloud/services/470928779762105>, 2018.
- [89] George Rupert Mason, Radu Constantin Calinescu, Daniel Kudenko, and Alec Banks. Assured reinforcement learning for safety-critical applications. In *Doctoral Consortium at the 10th International Conference on Agents and Artificial Intelligence*. SciTePress, 2017.
- [90] George Rupert Mason, Radu Constantin Calinescu, Daniel Kudenko, and Alec Banks. Assured reinforcement learning with formally verified abstract policies. In *9th International Conference on Agents and Artificial Intelligence (ICAART)*. York, 2017.
- [91] Ayhan Mehmed, Sasikumar Punnekkat, Wilfried Steiner, Giacomo Spampinato, and Martin Lettner. Improving dependability of vision-based advanced driver assistance systems using navigation data and checkpoint recognition. In *SAFECOMP 2015: Proceedings of the 34th International Conference on Computer Safety, Reliability, and Security*, volume 9337 of *Lecture Notes in Computer Science*, pages 59–73, Delft, The Netherlands, September 2015. Springer-Verlag.
- [92] CADE METZ. How Google’s AI viewed the move no human could understand. <https://www.wired.com/2016/03/googles-ai-viewed-move-no-human-understand/>, 2018.
- [93] Bertrand Meyer. On formalism in specifications. In *Program Verification*, pages 155–189. Springer, 1993.
- [94] Patrice Micouin. Toward a property based requirements theory: System requirements structured as a semilattice. *Systems engineering*, 11(3):235–245, 2008.
- [95] Patrice Micouin. *Model Based Systems Engineering: Fundamentals and Methods*. John Wiley & Sons, 2014.
- [96] David J. Miller, Yujia Wang, and George Kesidis. When not to classify: Anomaly detection of attacks (ADA) on DNN classifiers at test time. *arXiv preprint arxiv:1712:06646*, December 2017.
- [97] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.
- [98] Niklas Möller. The concepts of risk and safety. In *Handbook of risk theory*, pages 55–85. Springer, 2012.
- [99] Niklas Möller and Sven Ove Hansson. Principles of engineering safety: risk and uncertainty reduction. *Reliability Engineering & System Safety*, 93(6):798–805, 2008.

- [100] NASA. *Fault Management Handbook*. NASA-HDBK-1002, 2011.
- [101] Cu D. Nguyen, Simon Miles, Anna Perini, Paolo Tonella, Mark Harman, and Michael Luck. Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, 25(2):260–283, 2012.
- [102] Gregory M. Papadopoulos. Data flow models for fault-tolerant computation. Master’s thesis, Massachusetts Institute of Technology, June 1983.
- [103] Nicolas Papernot et al. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arxiv:1511.04508*, March 2016.
- [104] Requirements and Technical Concepts for Aviation (RTCA), Washington, DC. *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, December 2011.
- [105] David J. Rinehart, John C. Knight, and Jonathan Rowanhill. *Understanding what it Means for Assurance Cases to “work”*. National Aeronautics and Space Administration, Langley Research Center, 2017.
- [106] John Rushby. Verification diagrams revisited: Disjunctive invariants for easy verification. In *International Conference on Computer Aided Verification*, pages 508–520. Springer, 2000.
- [107] John Rushby. Modular certification. NASA Contractor Report CR-2002-212130, NASA Langley Research Center, December 2002. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/2002/cr/NASA-2002-cr212130.pdf>.
- [108] John Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167–177, February 2002.
- [109] John Rushby. Runtime certification. In Martin Leucker, editor, *Eighth Workshop on Runtime Verification: RV08*, volume 5289 of *Lecture Notes in Computer Science*, pages 21–35, Budapest, Hungary, April 2008. Springer-Verlag.
- [110] John Rushby. The versatile synchronous observer. In S. Iida, J. Meseguer, and K. Ogata, editors, *Specification, Algebra, and Software, A Festschrift Symposium in Honor of Kokichi Futatsugi*, volume 8373 of *Lecture Notes in Computer Science*, pages 110–128, Kanazawa, Japan, April 2014. Springer-Verlag.
- [111] John Rushby. On the interpretation of assurance case arguments. In *New Frontiers in Artificial Intelligence: JSAI-isAI 2015 Workshops, LENLS, JURISIN, AAA, HAT-MASH, TSDAA, ASD-HR, and SKL, Revised Selected Papers*, volume 10091 of *Lecture Notes in Artificial Intelligence*, pages 331–347, Kanagawa, Japan, November 2015. Springer-Verlag.
- [112] John Rushby. Assurance and assurance cases. In A. Pretschner, D. Peled, and T. Hutzelmann, editors, *Dependable Software Systems Engineering (Marktoberdorf Summer School Lectures, 2016)*, Volume 50 of NATO Science for Peace and Security Series D, pages 207–236. IOS Press, October 2017.

- [113] John Rushby. PVS embeddings of propositional and quantified modal logic. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, November 2017. Available at <http://www.csl.sri.com/users/rushby/papers/modalpvs.pdf>.
- [114] Aviation Safety. Accident description boeing 747-237b. Available at <https://aviation-safety.net/database/record.php?id=19780101-1>.
- [115] SAL home page. <http://sal.csl.sri.com/>.
- [116] Andrea Scarinci. *Monitoring safety during airline operations: A systems approach*. PhD thesis, Massachusetts Institute of Technology, 2017.
- [117] Matthias Scheutz, Scott A. DeLoach, and Julie A. Adams. A framework for developing and using shared mental models in human-agent teams. *Journal of Cognitive Engineering and Decision Making*, page 1555343416682891, 2017.
- [118] Alan C. Schultz, John J. Grefenstette, and Kenneth A. De Jong. Test and evaluation by genetic algorithms. *IEEE expert*, 8(5):9–14, 1993.
- [119] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.
- [120] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards verified artificial intelligence. *arXiv preprint arXiv:1606.08514*, 2016.
- [121] Lui Sha. Using simplicity to control complexity. *IEEE Software*, 18(4):20–28, July 2001.
- [122] Thomas B Sheridan and Raja Parasuraman. Human versus automation in responding to failures: An expected-value analysis. *Human factors*, 42(3):403–407, 2000.
- [123] Yasser Shoukry et al. Secure state estimation for cyber physical systems under sensor attacks: A satisfiability modulo theory approach. *IEEE Transactions on Automatic Control*, 62(10):4917–4932, October 2017.
- [124] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.
- [125] Gurjeet Singh. The trust challenge – Why explainable AI is not enough. <https://www.ayasdi.com/blog/artificial-intelligence/trust-challenge-explainable-ai-not-enough/>, 2018.
- [126] Ben Smith, Martin Feather, and Terry Huntsberger. A hybrid method of assurance cases and testing for improved confidence in autonomous space systems. In *AIAA Infotech@Aerospace Conference*, pages 1981–, Kissimmee, FL, January 2018.
- [127] Society of Automotive Engineers. *Aerospace Recommended Practice (ARP) 4754A: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, December 2010. Also issued as EUROCAE ED-79.

- [128] Open Group Standard and The Open Group. The TOGAF® Standard, Version 9.2. pages 1–181, 2013.
- [129] Wilfried Steiner, Ayhan Mehmed, and Sasikumar Punnekkat. Improving intelligent vehicle dependability by means of infrastructure-induced tests. In *Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference*, pages 147–152, 2015.
- [130] Lorenzo Strigini and Andrey Povyakalo. Software fault-freeness and reliability predictions. In *SAFECOMP 2013: Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security*, volume 8153 of *Lecture Notes in Computer Science*, pages 106–117, Toulouse, France, September 2013. Springer-Verlag.
- [131] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*, 2017.
- [132] John Thomas and Dajiang Suo. STPA-based method to identify and control feature interactions in large complex systems. *Procedia Engineering*, 128:12–14, 2015.
- [133] John P Thomas IV. *Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis*. PhD thesis, Massachusetts Institute of Technology, 2013.
- [134] Ashish Tiwari, Bruno Dutertre, Dejan Jovanović, Thomas de Candia, Patrick D. Lincoln, John Rushby, Dorsa Sadigh, and Sanjit Seshia. Safety envelope for security. In *Proceedings of the 3rd International Conference on High Confidence Networked Systems (HiCoNS)*, pages 85–94, Berlin, Germany, April 2014. ACM.
- [135] Perry Van Wesel and Alwyn E. Goodloe. Challenges in the verification of reinforcement learning algorithms. Technical report, NASA, 2017.
- [136] Kush R. Varshney and Homa Alemzadeh. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big Data*, 5(3):246–255, 2017.
- [137] Matt Webster, Michael Fisher, Neil Cameron, and Mike Jump. Formal methods for the certification of autonomous unmanned aircraft systems. In *SAFECOMP 2011: Proceedings of the 30th International Conference on Computer Safety, Reliability, and Security*, volume 6894 of *Lecture Notes in Computer Science*, pages 228–242, Naples, Italy, September 2011. Springer-Verlag.
- [138] John H. Wensley, Leslie Lamport, Jack Goldberg, Milton W. Green, Karl N. Levitt, P. M. Melliar-Smith, Robert E. Shostak, and Charles B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.
- [139] Stephen Whitlow, Chris Wilkinson, and Chris Hamblin. Assessing v and v processes for automation with respect to vulnerabilities to loss of airplane state awareness. 2014.
- [140] Casimir Wierzynski. The challenges and opportunities of explainable ai. <https://ai.intel.com/the-challenges-and-opportunities-of-explainable-ai/>, 2018.

- [141] Chris Wilkinson, Brendan Hall, Kevin Driscoll, Nancy G. Leveson, Paul Snyder, and Frank McCormick. Identify safety issues in integration of complex digital systems. NASA Contractor Report DOT/FAA/AR-xx/xx, NASA Langley Research Center, Hampton, VA, January 2018.
- [142] Chiyuan Zhang et al. Understanding deep learning requires rethinking generalization. *arXiv preprint arxiv:1611.03530*, February 2017.
- [143] Xingyu Zhao, Bev Littlewood, Andrey Povyakalo, Lorenzo Strigini, and David Wright. Modeling the probability of failure on demand (pfd) of a 1-out-of-2 system in which one channel is “quasi-perfect”. In *Reliability Engineering and System Safety*, pages 230–245. Elsevier Ltd, February 2017.

A An Examination of Agent Safety Logic

It is reasonable to assume that human pilots’ interactions with an aircraft and its systems—and with the larger environment consisting of other aircraft, air traffic control, and the world at large—are driven by their knowledge and beliefs about those elements and systems. Knowledge and belief are closely related: knowledge has traditionally been characterized as justified true belief, although modern treatments challenge this and set a high bar for justification. We discuss this topic from the assurance perspective elsewhere [112]; here, we are interested in using logical properties of belief and knowledge to model interactions between pilots and cockpit automation.

Belief and knowledge are *modes* of truth: it is one thing for a proposition to be *true*, and another to *believe* that it is true, or to *know* that it is true. Modal logics are logics that are specialized for reasoning about such concepts and they can be provided with mechanized support by embedding them in the logic of a formal verification system. We have done this for both propositional and quantified modal logics using the PVS verification system and have provided a tutorial description of the method and its application [113].

Our goal here is to explore application of mechanized modal logics to modeling and analysis of some aspects of pilot-automation interaction. Rather than invent our own method, for this initial investigation our approach is to reproduce a method that uses an “Agent Safety Logic” developed by Seth Ahrenbach [3] that was subsequently mechanized using the Coq verification system [4]. We describe this activity in the following section and then conclude with a section that provides a critical discussion of the approach and suggests topics for further investigation.

Agent Safety Logic in PVS

We assume familiarity with the PVS embedding of modal logics [113]. Here, we will employ a shallow embedding for propositional modal logics as described in [113]; that prior description presented a single modal qualifier \Box (and its dual \Diamond) and explained how it could be given different interpretations (e.g., as knowledge, or as belief) by employing standard collections of axioms (e.g., knowledge corresponds to the standard axioms T45 and belief to D45) and that these correspond to algebraic properties of the access relation (e.g., an equivalence relation for knowledge, and serial, transitive, and Euclidean for belief).

Ahrenbach and Goodloe’s Agent Safety Logic [5] employs both knowledge and belief, so one approach would be to import two copies of the standard PVS shallow embedding and instantiate one for knowledge and the other for belief. However, Ahrenbach and Goodloe also use a modal qualifier to represent actions of the system, and reason about interactions among the qualifiers (e.g., belief about knowledge), so it seems more appropriate to construct a customized PVS theory, but one that is based directly on the standard shallow embedding of [113, Figure 8]. We show the beginning of this customized theory in Figure 24. The first dozen lines are exactly as in the standard embedding. We then introduce two accessibility relations on possible worlds; one, `accessK`, is for the knowledge

<pre> prop_safety_logic: THEORY BEGIN worlds: TYPE+ pvars: TYPE+ val: [pvars -> [worlds -> bool]] pmlformulas: TYPE = [worlds -> bool] v, w: VAR worlds x, y: VAR pvars P, Q: VAR pmlformulas ~(P)(w): bool = NOT P(w) ; &(P, Q)(w): bool = P(w) AND Q(w) ; =>(P, Q)(w): bool = P(w) IMPLIES Q(w) ; accessK: pred[[worlds,worlds]] accessB: pred[[worlds,worlds]] B(P)(w): bool = FORALL v: accessB(w, v) IMPLIES P(v) ; K(P)(w): bool = FORALL v: accessK(w, v) IMPLIES P(v) ; =(w, P): bool = P(w) valid(P): bool = FORALL w: w = P validval(x: pvars): boolean = valid(val(x)) CONVERSION valid, validval, val </pre>	PVS fragment
---	--------------

Figure 24: Specification of Ahrenbach and Goodloe's Basic Theory in PVS

modality and the other, `accessB`, is for belief. We then define the knowledge and belief modalities `K` and `B` themselves. Unlike \Box , there is no special syntax for `K` and `B`, so we write `K(P)` to indicate knowledge of `P` and similarly `B(P)` for belief in `P`. The remainder of the specification is the same as in the standard embedding.

Next, we define the axioms for `K` and `B`. The standard axioms for knowledge or *Epistemic* logic are T45 (also known as S5); axiom K (distribution, not to be confused with the modal qualifier `K`) is often mentioned as well but this is a theorem, true in all modal logics and easily derived when needed, so we omit it. Ahrenbach and Goodloe employ an unusual axiomatization for knowledge that omits standard axiom 5, which is the axiom for negative introspection $\sim K(P) \Rightarrow K(\sim K(P))$: if you do not

<pre> A2: AXIOM K(P) => P % standard axiom T, reflexive A3: AXIOM K(P) => K(K(P)) % standard axiom 4, transitive A5: AXIOM B(P) => ~B(~P) % standard axiom D, serial A6: AXIOM B(P) => B(B(P)) % standard axiom 4, transitive A7: AXIOM ~B(P) => B(~B(P)) % standard axiom 5, Euclidean A8: AXIOM K(P) => B(P) % Hintikka's axiom L1 A9: AXIOM B(P) => B(K(P)) % Variant of Hintikka's axiom L2 L1: LEMMA B(K(P)) => ~K(~K(P)) </pre>	PVS fragment
--	--------------

Figure 25: Specification of Ahrenbach and Goodloe’s Axioms in PVS

know P , then you know that you do not know it. We discuss choice of axioms for K in Section A.

Ahrenbach and Goodloe do use the standard axiomatization for belief or *Doxastic* logic: D45. The crucial difference between belief and knowledge is that belief substitutes standard axiom D for T; that is, if we know something, then it is true (standard axiom T), whereas if we merely believe it, all we can say is that we do not also believe its negation (standard axiom D).

Ahrenbach and Goodloe’s axiomatization is shown in Figure 25, where the PVS axioms are named after the corresponding axiom from [5] and their standard names are given in comments (Ahrenbach and Goodloe’s axioms 1 and 4 are omitted as they are distribution for K and B respectively, and are derivable). The comments also state properties of the accessibility relation that correspond to the axioms concerned; for example, axiom A6 corresponds to transitivity of `accessB`. These are standard results of modal logic and are proved in [113]. Incidentally, standard axiom D is usually written as $\Box P \supset \Diamond P$ where \Diamond is the dual modality, but we have not defined the duals for K or B , so we replace it by its definition $\neg\Box\neg$; hence, standard axiom D for B takes the form shown as A5. Similarly, standard axiom 5 is usually written $\Diamond Q \supset \Box\Diamond Q$, which becomes $\neg\Box\neg Q \supset \Box\neg\Box\neg Q$ and simplifies to $\neg\Box P \supset \Box\neg\Box P$ on replacing $\neg Q$ by P ; hence standard axiom 5 for B takes the form shown as A7.

Next, we give two axioms for the interaction of B and K . Axiom A8 simply says that you believe what you know, which Ahrenbach and Goodloe refer to as “Epistemic Principle 1”. It corresponds to axiom L1 in Hintikka’s classical treatment [67]. Axiom A9 says that if you believe P then you believe that you know P , which Ahrenbach and Goodloe refer to as “justified belief”. A9 reverses Hintikka’s classical treatment, where $B(P) \Rightarrow K(B(P))$ is his axiom L2 [67].

Finally, we derive Lemma L1, which Ahrenbach and Goodloe refer to as “Epistemic Principle 2”. It is proved with the following commands.

(lemma 'A5') (lemma 'A8') (grind :polarity? t)	PVS proof
--	-----------

In the flight context, expressions such as $K(P)$ and $B(P)$ indicate what pilots know and believe; we now need a way to indicate the *actions* that they perform. Ahrenbach and Goodloe provide a couple of pages of motivation and subsidiary definitions, but the construction that they add to their logic is a parameterized modal operator that they write as $\chi S\alpha$ and we write as $S(P, Q)$ where α (resp. Q) is represents an “action” and χ (resp. P) are conditions under which it is safe to perform α .

For the semantics of this operator, they specify

$$w \models \chi S\alpha \text{ iff } w \models \chi \wedge \exists v, w(\text{Safety})v \wedge v \models \alpha, \quad (2)$$

where *Safety* is a relation on worlds that is defined in terms of some subsidiary constructions.¹⁵

Both χ and α are modeled as propositions on the world, so in PVS we will represent S as a modal operator that takes two arguments, $S(P, Q)$, as follows.

Safety: pred[[worlds, worlds]] S(P,Q)(w): bool = P(w) and (EXISTS v: Safety(w, v) and Q(v))	PVS fragment
--	--------------

Here, we introduce *Safety* as an uninterpreted relation on worlds, then define S using exactly the formulation (2) from [5] given above: $S(P, Q)$ is true in world w if P is true in w and there is some *Safety*-accessible world v where Q is true.

Ahrenbach and Goodloe then introduce two lemmas, an axiom, and a theorem, which we transliterate directly as follows.

L2: LEMMA S(P,Q) => P A10: AXIOM Q => B(S(P,Q)) L3: LEMMA Q => B(K(S(P,Q))) T4: THEOREM Q => B(K(P))	PVS fragment
---	--------------

¹⁵It is possible that *Safety* is intended as the accessibility relation on S but this is not used anywhere, nor do the subsidiary constructions of Ahrenbach and Goodloe (which occupy three columns of text) play any part in the actual axiomatization, so we ignore them here.

Lemma L2 (their Lemma 2) is immediate from the definition of S (in PVS, it is proved by the single command (grind)). Ahrenbach and Goodloe refer to axiom A10 (their Axiom 10) as “Basic Pilot Safety” and state that a “basically safe pilot engages in action Q only if he believes . . . that the action is safe.” We consider this interpretation of the formula rather contentious: there is nothing that identifies Q as an action, so this axiom can be read as a claim about arbitrary modal propositions P and Q. Lemma L3 simply applies axiom A9 to A10. The proof in PVS is longer than it should be because PVS is poor at quantifier instantiation.

<pre>(lemma 'A9') (lemma 'A10') (grind :polarity? t :exclude ('S')) (repeat (inst?)) (grind :polarity? t))</pre>	PVS proof
--	-----------

Ahrenbach and Goodloe refer to lemma L3 as “pilot rationality” and interpret it to assert that a rational pilot performs an action only if the pilot believes that it is safe.

Intuitively, theorem T4 (their Theorem 4) simply combines lemmas L2 and L3, but it is a rather strange result as there is nothing to connect P and Q: it says that if a proposition P is true, then the pilot believes he or she knows some unrelated proposition Q. Once again, the PVS proof is longer than it should be due to extra steps necessitated by weak quantifier instantiation.

<pre>(lemma 'L3') (lemma 'L2') (grind :polarity? t :exclude ('S' 'B' 'K')) (inst -2 'P!1' 'Q!1' 'w!1') (grind)</pre>	PVS proof
--	-----------

Ahrenbach and Goodloe use this modeling framework to analyze some examples of pilot error. The first is Air India Flight 855, where the captain’s attitude indicator incorrectly indicated a right bank, leading him to command a steep left bank and thereby crash the aircraft. Ahrenbach and Goodloe present this as follows, where we substitute our variable and formulas names for theirs, and make a technical adjustment that is described later.

Let `all_say_right` be the proposition “all artificial horizons indicate a steep right bank” and let `here` be the current (real) world. Then `all_say_right` is false in world `here` because only the captain’s instrument indicated a right bank. Thus, we have `here |= ~all_say_right` and then, by A2, `here |= ~K(all_say_right)`. Let `left_bank` be the action “the pilot commands a steep left bank.” The pilot performs this action in the current world and so we have `here |= left_bank`. By T4, rational pilots perform an action only if they believe they knows it is safe: `here |=`

$B(K(\text{all_say_right}))$ and by L1 it then follows that here $\models \sim K(\sim K(\text{all_say_right}))$. Conjoining this with the already established here $\models \sim K(\text{all_say_right})$, propositional calculus gives us here $\models \sim(\sim K(\text{all_say_right}) \Rightarrow K(\sim K(\text{all_say_right})))$, which is the negation, or failure, of negative introspection for `all_say_right` in the current world.

We can formally specify and verify this in PVS as follows.

<pre> here: worlds all_say_right, left_bank: pmlformulas setup: AXIOM here $\models \sim \text{all_say_right} \ \& \ \text{left_bank}$ C1: LEMMA here $\models \sim K(\text{all_say_right})$ C2: LEMMA here $\models \sim K(\sim K(\text{all_say_right}))$ NI(P): pmlformulas = $\sim K(P) \Rightarrow K(\sim K(P))$ % negative introspection Failure: THEOREM here $\models \sim \text{NI}(\text{all_say_right})$ </pre>	PVS fragment
---	--------------

The axiom `setup` asserts that `all_say_right` is false and action `left.bank` is performed in the current world, denoted `here`. C1 and C2 correspond to the main deductions in the previous narrative. The first of these is proved by applying A2 to `setup`

<pre>(lemma 'setup') (lemma 'A2') (grind)</pre>	PVS proof
---	-----------

while the second applies T4 and L1, plus some manual proofs steps to guide the quantifier instantiation.

<pre>(lemma 'setup') (lemma 'T4') (lemma 'L1') (ground) (inst? -1) (inst?) (grind :polarity? t))</pre>	PVS proof
--	-----------

The modal predicate `NI(P)` defines negative introspection for `P` and the theorem `Failure` states that this fails for `all_say_right` in the current world. It is proved directly from the previous two lemmas.

```
(lemma 'C1') (lemma 'C2') (grind)
```

PVS proof

It is worth knowing what elements of the model are actually used in verifying this result, and the PVS Proofchain analysis provides this information, as shown below.

```
prop_safety_logic.Failure has been PROVED.
```

PVS proofchain

```
The proof chain for Failure is COMPLETE.
```

```
Failure depends on the following proved theorems:
```

```
prop_safety_logic.C1  
prop_safety_logic.C2  
prop_safety_logic.L1  
prop_safety_logic.L2  
prop_safety_logic.L3  
prop_safety_logic.T4
```

```
Failure depends on the following axioms:
```

```
prop_safety_logic.A10  
prop_safety_logic.A2  
prop_safety_logic.A5  
prop_safety_logic.A8  
prop_safety_logic.A9  
prop_safety_logic.setup
```

```
Failure depends on the following definitions:
```

```
prop_safety_logic.&  
prop_safety_logic.=>  
prop_safety_logic.B  
prop_safety_logic.K  
prop_safety_logic.NI  
prop_safety_logic.S  
prop_safety_logic.valid  
prop_safety_logic. |=  
prop_safety_logic.~
```

Now although PVS assures us that this development and its conclusion are valid, we find it rather unconvincing. First, there is nothing to connect `all_say_right` and `left_bank` to their intended

interpretations, nor to each other. The development would have proceeded identically if instead of `all_say_right` we had chosen some other proposition, such as “today is Christmas Day” (provided it is false in the then-current world), and we would then attribute the crash to failure of negative introspection on the date of Christmas. All applications of formal logic require both care and goodwill when attaching real-world interpretations to formal symbols and Ahrenbach and Goodloe provide careful motivation and description for those they intend, but we contend that their actual formalization is too weakly constrained and thereby admits too many unintended interpretations to serve as a guarantor of credible conclusions.

Second, lack of negative introspection is not the only, nor the most perspicuous, explanation for the Air India 855 scenario within Ahrenbach and Goodloe’s framework. Intuitively, it seems the captain acted on a false belief (that the aircraft had banked right), and we can formally verify this intuition. In the following, `FB(P)` defines `P` to be a false belief (i.e., one that is believed but untrue) and `Failure2`, in analogy to the earlier theorem `Failure`, asserts that the pilot held a false belief concerning `all_say_right`.

<pre>FB(P): pmlformulas = B(P) & ~P Failure2: THEOREM here = FB(all_say_right)</pre>	PVS fragment
--	--------------

This theorem is proved as follows.

<pre>(lemma 'setup') (lemma 'T4') (inst -1 'all_say_right' 'left_bank') (grind :polarity? t :exclude 'K') (use 'A2') (expand 'valid') (inst?) (ground)</pre>	PVS proof
--	-----------

Ahrenbach and Goodloe consider several other scenarios (Birgenair 301, Aeroperu 603, and Air France 447) but they all follow exactly the same pattern, which they state in their Theorem 5. We do not reproduce these developments here because they simply repeat the previous construction, with the propositions reinterpreted for the different scenarios, or generalized for Theorem 5, but the reasoning is unchanged.

One technical detail worth noting is that in their exposition of the Air India 855 example, Ahrenbach and Goodloe assert the validity of various modal formulas such as (in our notation) “the pilot believes he knows the relevant data: `B(K(all_say_right))`.” Validity means true in *all* worlds, whereas

these formulas are true only in the current world, so the correct assertions are of the form here $\models B(K(\text{all_say_right}))$. We used the correct form in our account, and Ahrenbach and Goodloe do so in their Theorem 5, but the loose usage in their examples diverted our PVS investigations for some time.

In the following section, we consider other aspects of Ahrenbach and Goodloe’s framework and their general conclusions. In the section following that, we examine an alternative approach.

Discussion

Ahrenbach and Goodloe [5] have proposed a novel and interesting model for human-machine interactions based on modal logics for belief and knowledge. We have reproduced this model in PVS using a shallow embedding of modal logics; the proofs that we have performed are very simple and the main benefit of the exercise has been to gain a better understanding of Ahrenbach and Goodloe’s model.

It seems reasonable that pilots’ interactions with cockpit automation will be guided by their beliefs and knowledge about the system and the current circumstances of the flight. It seems reasonable, too, to distinguish their knowledge (e.g., of how the automation works) from their beliefs (e.g., about the current circumstances of flight, based on sensor readings). Thus, we are sympathetic to the idea that both modes of thought should be represented and one applied to some issues and the other to different ones. What Ahrenbach and Goodloe typically do, however, is apply both together, so we have $B(K(P))$: the pilot *believes* that he or she *knows* some proposition P . We find it difficult to see intrinsic value in this combination: why $B(K(P))$ rather than simply $B(P)$: the pilot believes P ? Most models of artificial and natural intelligence are based on beliefs, desires, and intentions (BDI) and it seems unattractive to complicate this with beliefs about knowledge. Furthermore, knowledge is a very rare commodity—if we know something then it has to be true—and there is very little in an aircraft and its environment that a pilot can truly know, so rather than believe that he or she knows something, why not just believe it?

The objections above are philosophical; we also have technical criticisms. Modal logics for belief and knowledge are quite subtle; using the two in combination raises additional subtleties in their interaction. Thus, Ahrenbach and Goodloe’s axiom that we call A9 ($B(P) \Rightarrow B(K(P))$): if you believe something then you believe that you know it) is nonobvious but crucial to their accident analyses. Furthermore, it is the reverse of the standard axiom, namely Hintikka’s L2: $B(K(P)) \Rightarrow B(P)$, so this choice surely needs more motivation and justification.

Ahrenbach and Goodloe’s axiomatization of knowledge is also nonstandard: it omits standard axiom 5 for knowledge, which corresponds to “negative introspection” (if you do not know something then you know that you do not know it). Since their accident analyses lay the blame on lack of negative introspection, this looks contrived, to say the least. Adding standard axiom 5 for K to the framework would obviously eliminate Ahrenbach and Goodloe’s attribution of the cause of the four accidents to its lack, but the alternative attribution to false beliefs would be preserved. However, if we add standard axiom 5 for K to the framework, then we can derive $B(K(P)) \Rightarrow K(P)$, as specified below.

L1a: LEMMA $\sim K(P) \Rightarrow K(\sim K(P))$ IMPLIES $B(K(P)) \Rightarrow K(P)$

PVS fragment

This is proved as follows.

(lemma 'L1') (grind :polarity? t)

PVS proof

If we combine L1a with A9 we obtain $B(P) \Rightarrow K(P)$; taken together with A8, this indicates that knowledge and belief are essentially equivalent under this axiomatization if (contrary to Ahrenbach.et.al) we were to assume standard axiom 5 for K. The blame for this unfortunate outcome seems to be the nonstandard choice for A9.

Some logics of knowledge adopt a weakened version of standard axiom 5, known as axiom 4.4, where negative introspection is required only for true propositions.

A4_4: AXIOM $P \Rightarrow (\sim K(P) \Rightarrow K(\sim K(P)))$

PVS fragment

This might allow Ahrenbach and Goodloe to preserve their result with less appearance of contrivance. We have not examined this possibility.

For in addition to appearing contrived, we consider that attributing all four accidents to lack of negative introspection conflates wildly different scenarios and leads to the—in our opinion, flawed—recommendation [5] that enhanced negative introspection would be the way to prevent them in future.

In particular, Birgenair 301 and Aeroperu 603 both took off with faulty instruments (the static ports of Aeroperu 603 were covered by maintenance tape and Birgenair 301 had a pitot tube blocked by a wasps nest). There was no lack of negative introspection, in both cases the pilots knew they had faulty instruments, but they were simply unable to safely fly the aircraft in the (uncertified) conditions where they lacked air data.

In the case of Air India 855, the pilot flying had a false belief (based on a faulty attitude indicator) and acted on it; this is not a lack of negative introspection, but a failure to adequately question a belief reached in haste.¹⁶

In the case of Air France 447, the pitot tubes were blocked by ice crystals and the pilot flying commanded and maintained a high angle of attack, resulting in a stall. One hypothesis is that the pilot flying reverted to low-altitude recovery where stall protections allow full deflection of the sidestick to deliver maximum climb performance. But with blocked pitot tubes, the aircraft had entered an

¹⁶The pilot flying said that his attitude indicator had “toppled” (because it [incorrectly] indicated a right bank); the pilot monitoring said that his had toppled, too (because it was indicating the hard left bank commanded by the pilot flying). Several other instruments were available to the pilot flying, but not consulted, despite attempts by the flight engineer to direct his attention to them.

alternate mode where stall protections are disabled. It is moot whether the pilot flying did not know the aircraft was in alternate mode, or knew it but did not know that it had the effect of removing stall protection. But it seems to us that in either case, the problem was the pilot flying's false belief that stall protection was active, not his lack of knowledge about his lack of knowledge (i.e., negative introspection) concerning that function.

In almost all cases where lack of knowledge or a false belief leads to inappropriate action, it seems possible to argue that negative introspection would save the day: if the pilot knew what he or she didn't know, alternative action would be taken. But this seems rather indirect: surely the most constructive behavior by aircraft automation would be to eliminate the source of false belief or to challenge the action itself, based on more complete consultation of instruments and a more rational assessment of belief. Aircraft have many redundant sensors and sources of information concerning aircraft state, but in the presence of failure it is often left to the pilot to detect disagreement among sources and to select the most reliable. But automated diagnosis would almost certainly perform this task better than human pilots. For Birgenair 301 and Aeroperu 603, automated diagnosis would surely have detected flawed air data early in the takeoff while there was plenty of time to abort it. Air India 855 had several sources of correct attitude information that could have been used to mask the failure of the captain's instrument. Air France 447 is a more complicated case but measures for preventing its recurrence typically focus on more robust automation (not tripping out at the first sign of difficulty) and more active and constructive (CRM-like) partnership between enhanced (IA) automation and human crew, rather than adjusting informational cues to counter absence of negative introspection.

In summary, it seems to us that although models of knowledge and belief seem plausible as an approach to representing and analyzing interactions between pilots and cockpit automation, the approach described here is not promising. An alternative interpretation of interaction between pilots and automation is that it is driven by (imperfect) mental models [35, 59, 108]. An approach that may be worthy of consideration is to combine mental models (e.g., of how the pilot thinks things work) with beliefs (e.g., about the current state). We provide an initial exploration of this approach in the following section.

An Alternative Approach

In previous sections, we examined use of modal logics of knowledge and belief to model pilot behavior in human-computer interaction, following an approach proposed by Ahrenbach and Goodloe [5]. We noted that use of both knowledge and belief, with rather complex rules for their nested application and interaction (i.e., believing that you believe something, or believing that you know something), made for complicated and rather contrived models. Furthermore, choice of modal axioms can be contentious and can also appear contrived. For example, logics of knowledge often include the standard modal axiom known as 5, which models negative introspection (you know what you do not know), whereas the models of Ahrenbach and Goodloe exclude it and then attribute several aircraft crashes to its lack.

We noted that an alternative interpretation of interaction between pilots and automation is that it is driven by (imperfect) mental models [35,59,108]. We suggested that an approach that may be worthy of consideration is to combine mental models (e.g., of how the pilot thinks things work) with beliefs (e.g., about the current state). A potential difficulty is that systems and mental models are typically specified as state machines, while beliefs are interpreted in “possible worlds” and are represented as state machines of a different kind. Some of this difficulty can be seen in Ahrenbach and Goodloe’s model where actions are represented as modal operators in order to fit the same modeling framework as belief and knowledge. A possible way around this difficulty is to represent beliefs in a possible worlds model that is separate from the state machines for the system and the mental model, but interacts with them, with overall evaluation of safety properties being performed by a synchronous observer [110].

Here, we describe a small example that explores the feasibility of this approach using model checking in SAL [115]. We use a simple example based on a trivialization of the Air France 447 scenario [27]. The purpose of this example is not accurate modeling but to explore feasibility of a certain approach to modeling. We begin by defining five scalar types that are used to record the state of the aircraft.

```
pilotlogic: CONTEXT =  
BEGIN  
  
autopilot_state: TYPE = {engaged, disengaged};  
pitot_state: TYPE = {clear, blocked};  
protection_state: TYPE = {enabled, disabled};  
attitudes: TYPE = {level, dive, stall};  
stick_state: TYPE = {forward, neutral, back};
```

SAL fragment

The first three types specified here concern autopilot engagement, the state of the pitot tubes, and whether or not stall protections are enabled, respectively. The fourth abstracts the attitude of the aircraft: whether it is in level flight, diving, or stalled. Of course, there are many other possible attitudes, but these are sufficient for our limited purpose. Similarly, the final type is used to abstract the sidestick position of the pilot flying: whether it is pushed forward, in the neutral position, or pulled back.

Next we describe the behavior of the aircraft; we begin with its input and output variables.


```
aircraft: MODULE =  
BEGIN  
  INPUT  
    stick: stick_state  
  OUTPUT  
    autopilot: autopilot_state,  
    pitots: pitot_state,  
    protection: protection_state,  
    instrument_conflict: boolean,  
    attitude: attitudes  
  INITIALIZATION  
    autopilot = engaged;  
    pitots = clear;  
    instrument_conflict = FALSE;  
    protection = enabled;  
    attitude = level;
```

The only input variable is `stick`, whose value is the `stick_state` commanded by the pilot flying. The outputs (and, implicitly, state variables) are those recording the states of the autopilot, pitots, and stall protections, whether there is a conflict among the instruments, and the attitude of the aircraft. The output variables are then assigned their initial values.

Next, we specify the state transitions of the aircraft model.

```
TRANSITION
[
blocked_pitots:
  pitots = clear -->
    pitots' = blocked;
    autopilot' = disengaged;
    protection' = disabled;
[]
conflict:
  pitots = blocked --> instrument_conflict' = TRUE;
[]
unprotected_climb:
  stick = back AND protection = disabled --> attitude' = stall;
[]
other:
  TRUE -->
]
END;
```

We distinguish only three significant cases; all others are abstracted into the **other** “do nothing” case. Note that the semantics of SAL are nondeterministically to select and execute any one of the actions whose guard is true.

The **blocked_pitots** case applies when previously clear pitots become blocked; in this case the autopilot disengages and the stall protections are disabled. (Stall protections actually are related to whether primary flight control is in the normal or alternate mode, but we do not model this level of detail.)

When the pitots are blocked, it is possible that instruments reliant on air data will conflict with one another; this is modeled by the **conflict** case.

Finally, if protections are disabled and the pilot flying pulls back the sidestick, it is possible that a full stall will ensue; this is modeled by the **unprotected_climb** case.

Next, we describe the behavior of the pilot flying. We begin with the input and output variables.

```
pilot: MODULE =
BEGIN
  INPUT
    autopilot: autopilot_state,
    instrument_conflict: boolean
  LOCAL
    autopilot_belief: autopilot_state
  OUTPUT
    stick: stick_state,
    believes_protected, believes_unprotected: boolean
  INITIALIZATION
    stick = neutral;
    autopilot_belief = engaged;
    believes_protected = TRUE;
    believes_unprotected = FALSE;
```

One of input variables is the actual state of the autopilot (i.e., whether it is engaged or not). The pilot can learn this by observing an indication on the display, but may fail to do so and instead infer it by aircraft behavior or other cues. Consequently, we model the pilot's awareness of this state as a belief that may or may not accurately track the true state. This belief is recorded in the local state variable `autopilot_belief`.

The other input is `instrument_conflict` and we could treat the pilot's interpretation of this as a belief in the same way as `autopilot_state` but, for variety, we assume that the pilot always correctly discerns this and therefore use the actual state value as the pilot's interpretation.

The only true output generated by the pilot is the `stick_state`. The remaining state variables are used to model the pilot's beliefs about stall protection. Conceptually, these are local variables, but we will be using a synchronous observer to manage them and need to specify them as output variables so that the observer can read them.

Whereas `autopilot_belief` is modeled as a state variable of the same type as `autopilot` that may or may not accurately track its value, we model beliefs about stall protection in a different way that more closely resembles that used in logics of belief. We do this for variety and for illustration. In modal logics of belief, we can apply a belief operator B to any proposition. In particular, we may have beliefs about a proposition p , $B(p)$ and about its negation $B(\text{NOT } p)$. One of the defining characteristics of belief as a modality is the modal axiom known as D , which asserts that you cannot believe both a proposition and its negation. We will allow the pilot to hold separate beliefs about engagement and disengagement of the stall protections and we will use a synchronous observer to ensure that these obey the requirements of Axiom D .

The transitions of the pilot model are specified as follows.

```

TRANSITION
[
note_autopilot_off:
  autopilot = disengaged -->
    autopilot_belief' = disengaged;
    believes_protected' IN {believes_protected, FALSE};
    believes_unprotected' IN {TRUE, FALSE};
[]
note_autopilot_on:
  autopilot = engaged -->
    autopilot_belief' = engaged;
    believes_protected' = TRUE
[]
whoops:
  believes_protected
  AND autopilot_belief = disengaged
  AND instrument_conflict -->
    stick' = back
[]
idle:
  TRUE -->
]
END;

```

As with the aircraft model, we distinguish only three significant cases; all others are abstracted into the idle “do nothing” case.

The first two cases represent how pilots track the state of the autopilot. The first is the case where pilots observe or infer that the autopilot is disengaged. In this case, pilots update beliefs about the autopilot state appropriately. Pilots also update beliefs about stall protections, but do so in a possibly inconsistent manner. The pilots’ belief that protection is enabled is nondeterministically left unchanged, or changed to FALSE, whereas belief that protection is disabled is set in a completely nondeterministic manner. Notice that the autopilot can disengage without this rule being invoked, because the `idle` rule has the guard TRUE and can be selected in preference to this one.

The second rule applies when pilots observe or infer that the autopilot is engaged; here pilots update beliefs about the autopilot state appropriately and sets beliefs that stall protection is enabled to TRUE (belief that stall protection is disabled is unchanged). Finally, if the instruments conflict and the pilot believes the autopilot is disengaged but that stall protection is enabled, the pilot pulls back on the sidestick.

As noted, the pilot's beliefs about stall protection are modeled as somewhat erratic and inconsistent. At the very least we should require these beliefs to be consistent with modal Axiom D, so that the pilot cannot believe that protection is both enabled and disabled.

We use a synchronous observer for this purpose [110].

```
observer: MODULE =
BEGIN
  INPUT
    believes_protected, believes_unprotected: boolean
  OUTPUT
    aok: boolean
  INITIALIZATION
    aok = TRUE
  TRANSITION
    aok' = aok AND (believes_protected' XOR believes_unprotected')
END;
```

SAL fragment

The observer module takes the state variables representing beliefs about stall protection as input and sets the variable `aok` to `FALSE` if these ever become inconsistent (i.e., if neither or both of them become `TRUE`). Notice that the transition responds to the updated (i.e., primed) values of the belief variables.

We then assemble a complete system as the synchronous composition of the `aircraft`, `pilot`, and `observer` modules.

```
system: MODULE = aircraft || pilot || observer;
```

SAL fragment

Finally, we ask if there is ever a case when the aircraft attitude enters the `stall` state while `aok` is `TRUE` (i.e., without there being an inconsistency in beliefs about stall protection). We assert the negation of this as an invariant.

```
test: THEOREM system |- G(aok => NOT attitude = stall);
```

SAL fragment

We then invoke the SAL model checker (we use bounded model checking with iterative deepening, the symbolic model checker will do equally well).

```
sal-bmc pilotlogic.sal test -it --delta-path
```

SAL command

This finds the following 4-step counterexample. Only those state variables whose values have changed are displayed in the intermediate steps.

```
=====
Path
=====
Step 0:
--- System Variables (assignments) ---
aok = true
attitude = level
autopilot = engaged
autopilot_belief = engaged
believes_protected = true
believes_unprotected = false
instrument_conflict = false
pitots = clear
protection = enabled
stick = neutral
-----
Transition Information:
  label blocked_pitots
  label idle
-----
Step 1:
--- System Variables (assignments) ---
autopilot = disengaged
pitots = blocked
protection = disabled
-----
Transition Information:
  label conflict
  label note_autopilot_off
-----
Step 2:
--- System Variables (assignments) ---
autopilot_belief = disengaged
instrument_conflict = true
-----
Transition Information:
  label other
  label whoops
-----
```

Continued...

```
...continued

Step 3:
--- System Variables (assignments) ---
stick = back

Transition Information:
  label unprotected_climb
  label note_autopilot_off
-----

Step 4:
--- System Variables (assignments) ---
aok = true
attitude = stall
autopilot = disengaged
autopilot_belief = disengaged
believes_protected = false
believes_unprotected = true
instrument_conflict = true
pitots = blocked
protection = disabled
stick = back
```

In the scenario discovered by the model checker, the pitots become blocked, causing the autopilot to become disengaged and stall protection to be disabled. Next, the pilot notes the autopilot is disengaged and updates beliefs appropriately. The instruments also display conflicting information but the pilot still believes that stall protection is enabled, so he or she pulls back on the sidestick, causing an unprotected climb and stall.

Summary

We described our objections to Ahrenbach and Goodloe's agent safety logic [5] in Section A. Briefly, we find their choice of axioms contentious, and also their attribution of the accidents considered to lack of negative introspection. We agree that models of belief and knowledge could be useful elements in the analysis of cockpit automation, but we prefer their direct application to the second-level forms (i.e., knowing and believing what you know and don't know) employed by Ahrenbach and Goodloe.

We also consider modal logic a rather unsatisfactory vehicle for this type of modeling, as it requires the system model to be represented in the same way. State machines provide a more natural system model, so we explored whether reasoning about knowledge and belief can be applied in a state

machine model.

We constructed an elementary model of interaction between pilot and aircraft based on the pilot's beliefs about aspects of the system state, and applied it to a trivialized scenario based on AF 447.

Unlike the modal logic models of Ahrenbach, this model has no systematic treatment of belief: we simply represent beliefs as independent state variables that track the corresponding real system state with different levels of accuracy. We showed how some beliefs can be allowed to evolve independently but be required to stay in some relationship with each other (e.g., to satisfy modal axiom D) using a synchronous observer. We have no need for beliefs about beliefs, nor beliefs about knowledge. The resulting system and pilot models seem easy both to develop and to validate, and the dangerous scenarios found as counterexamples seem straightforward to interpret.

In the trivialized example developed here, there is no explicit mental model: it is implicit in the action *whoops* that it is permissible to pull back on the sidestick if protections are enabled. A more developed example would include an explicit mental model for safely hand-flying an aircraft at cruise altitude.

Our basic method for analysis of human-computer interaction by means of model checking proposed juxtaposition of an aircraft model and a possibly oversimplified or faulty “mental model” [59, 108]. The small amendment made here is to suppose that inputs to the mental model are “beliefs” about system state, rather than the state itself, and that beliefs may sometimes be inaccurate.

In conclusion, both approaches examined here are at a very preliminary stage of development and it not possible to recommend adoption or rejection of either: further research is needed. It does seem, however, that modeling and analysis of IA aircraft, and possibly construction of the IA functions themselves, will require explicit modeling of pilot beliefs and knowledge and that future research should indeed consider these topics.

B Requirements of IA Systems in Reduced Crew Operations

B.1 Information Requirements

Operational requirements have been identified in terms of the tasks, procedures and error-checking functions of the pilot flying (PF) and the pilot monitoring (PM) on the flight crew, with the expectation that the automated system will at any given time and context perform the roles of either PF, PM, a hybrid of the two, or both.

In terms of information requirements, the requirement language stipulates whether the information is static or continuously updated during flight, as indicated below:

Requirement language:	Meaning:
The system shall obtain. . .	changes before each flight or during flight
The system shall have knowledge of. . .	stored memory - supports all flights

Table 3: Information Updating

The information requirements identified for the IA system are listed below:

- The system shall have knowledge of Standard Operating Procedures (SOP) as well as fault and emergency procedures.
- The system shall have knowledge of airports, procedures and navigational aids.
- The system shall obtain operational phase of flight and shall perform Pilot Flying (PF) tasks and procedures within that phase, based on SOP.
- The system shall obtain operational phase of flight and shall perform Pilot Monitoring (PM) tasks and procedures within that phase, based on SOP.
- The system shall have knowledge of the impact of weather conditions on aircraft performance,
- The system shall obtain unexpected (unpredicted) changes in weather patterns.
- The system shall have knowledge of, and capability to modify its expectations of aircraft performance based on sensed weather conditions as appropriate.
- The system shall obtain and store ATC communications
- The system shall obtain notice when reaching a waypoint or fix
- The system shall obtain the Flight plan
- The system shall obtain FAA approvals
- The system shall obtain the Contingency flight plan
- The system shall have knowledge of Airspace information
- The system shall have knowledge of Chart data from all available sources
- The system shall have knowledge of Navigation Procedures
- The system shall have knowledge of Standard operating procedures for communications
- The system shall have knowledge of Standard operating procedures for functions

- The system shall have knowledge of Knowledge of the airline’s task management paradigm (SOP)
- The system shall have knowledge of When deviation from SOP should be expected
- The system shall have knowledge of Expected pilot actions for normal procedures
- The system shall have knowledge of Expected pilot actions based on ATC communications
- The system shall obtain notice when reaching transition point
- The system shall obtain notice when reaching decision height
- The system shall have knowledge of System functions - normal (range)
- The system shall Trigger alert when detecting unexpected system functions
- The system shall have knowledge of Control inputs (safe range)
- The system shall obtain notice when detecting control inputs outside safe range
- The system shall obtain the Latest FMS database
- The system shall obtain Traffic location, range, distance, speed and direction of movement
- The system shall obtain Traffic density (radar)
- The system shall obtain own aircraft current physical (GPS) location
- The system shall have knowledge of Aircraft state and when outside normal range
- The system shall have knowledge of Aircraft attitude and when outside normal range
- The system shall have knowledge of Terrain data
- The system shall have knowledge of Obstacle data
- The system shall have knowledge of Location of cities/densely populated areas
- The system shall have knowledge of Emergency transponder codes
- The system shall obtain Fuel level information
- The system shall have knowledge of emergency management techniques
- The system shall have knowledge of Alert priorities
- The system shall obtain radio frequencies based on the flight plan and ATC communication
- The system shall obtain the “Out the window view”
- The system shall obtain and store crew communications
- The system shall have knowledge of appropriate verbal response to crew communications
- The system shall obtain Radar frequencies based on the flight plan
- The system shall have knowledge of Safe speeds and operating speeds (climb/descent rates) unique to AC
- The system shall obtain Nature of emergency when detected by crew
- The system shall have knowledge of Emergency procedures (not already in checklist)
- The system shall have knowledge of Troubleshooting procedures
- The system shall obtain NOTAMs and airport/airspace updates & restrictions
- The system shall obtain up-to-date Weather data - AIRMET, SIGMET
- The system shall obtain Task progress and tasks in queue
- The system shall obtain Crew member task progress and tasks in queue

- The system shall have knowledge of Task pass/fail criteria
- The system shall have knowledge of aircraft state and orientation.
- The system shall have knowledge of aircraft subsystem states and modes.
- The system shall have knowledge of and expect normal aircraft responses to control inputs.
- The system shall be aware of the current flight plan and aircraft location on the flight plan.

B.2 Functional Requirements

Functional requirements for the automation to support the operational interface are presented below.

- The system shall continuously calculate height above terrain (AGL)
- The system shall trigger an alert when reaching a waypoint or fix
- The system shall trigger an alert when reaching transition point
- The system shall trigger an alert when reaching decision height
- The system shall trigger an alert when detecting unexpected system functions
- The system shall trigger an alert when detecting control inputs outside safe range
- The system shall review navigation chart data, select pertinent data and update/maintain FMS plan
- The system shall review NOTAM/PIREP data, select pertinent data and update/maintain FMS plan
- The system shall review weather data, select pertinent data and update/maintain FMS plan
- The system shall review airport data, select pertinent data and update/maintain FMS plan
- The system shall perform check of systems status displays for fault or failure conditions
- The system shall set new waypoint in flight plan
- The system shall check automation response and progress on course to new waypoint
- The system shall continuously check aircraft speed against expected/safe range for maneuver and flight phase
- The system shall continuously check aircraft altitude and vertical speed against cleared altitudes, climbs and descents
- The system shall continuously check aircraft bank rate against expected/safe range
- The system shall identify local navigation aids
- The system shall acknowledge need to climb/descend, select desired altitude
- the system shall recognize proper aircraft response and/or lack of timely response to control inputs
- The system shall acknowledge need to change heading, select desired heading
- The system shall check status of human crew (liveliness)
- The system shall check status of human crew (unusual actions)
- The system shall collect biometrics of human crew as appropriate
- The system shall Identify need & what to communicate to human crew

- The system shall Identify need & what to communicate to ATC
- The system shall set and verify radio frequency; backup frequencies
- The system shall set and verify navigation aid frequencies
- The system shall check human crew actions against expectations and SOP
- The system shall identify lack of crew response to communications within XX seconds
- The system shall acknowledge fault or failure, consult appropriate manuals/checklists, confer with crew mate and perform corrective action(s). Verify clearance of fault or perform next step.
- The system shall monitor aircraft progress on flight path or procedure, crew takes action as necessary to remain on flight path
- The system shall acknowledge need to refer to manual, find topic, perform calculation or address situation accordingly
- The system shall identify needs for calculation, consult reference or mentally calculate and set instruments/FMS accordingly
- The system shall continuously Monitor instruments and flight displays
- The system shall observe crossing fix or waypoint
- The system shall observe crossing altitude
- The system shall locate, access and interpret navigational charts
- The system shall visually search outside the aircraft
- The system shall check outside (visual) information against traffic displays (integrate info)
- The system shall Interface with and monitor flight controls
- The system shall check automation mode
- The system shall monitor automation modes and set new modes or switch to manual control as necessary
- The system shall disengage automation in the MCP
- The system shall fly “manually”
- The system shall perform aircraft maneuvers as appropriate
- The system shall enter new control mode as appropriate
- The system shall set automation mode in the MCP
- The system shall select engage autopilot
- The system shall monitor weather radar
- The system shall reference altitude and current temperature
- The system shall identify appropriate power; adjust power accordingly
- The system shall perform “mental calculations” and metric conversions as required
- The system shall compile and integrate resource(s) for weather, navigation, decision making
- The system shall review traffic display and determine if action is necessary
- The system shall locate appropriate checklists
- The system shall verbally read each sequential checklist task and wait for a response
- The system shall take corrective action as necessary for human crew to complete checklist

- The system shall perform checklist actions and indicate task completion or failure to complete task
- The system shall search visually outside the aircraft and weather radar (mentally integrate info)
- The system shall review the terrain data and determine if action is necessary
- The system shall take action to avoid terrain and notify crewmate as necessary
- The system shall dynamically adapt to abnormal context such as subsystem faults and failures
- The system shall inform the pilot of abnormal context such as subsystem faults and failures
- The system shall perform fault, failure and emergency procedures

B.3 Communications Requirements

Functional requirements for the automation to support the operational interface are presented below.

- The system shall accept and respond to crew communications
- The system shall accept and respond to ATC communications
- The system shall monitor/“listen” on the active radio frequency for aircraft tail number/callsign
- The system shall recognize aircraft callsign
- The system shall “listen” for further communication upon callsign recognition
- The system shall identify and log communication from ATC attributed to aircraft callsign
- The system shall provide/log confidence level for speech recognition accuracy
- The system shall alert human crew to communication (given lack of response)
- The system shall respond directly to ATC via voice
- The system shall respond directly to ATC via digital communication
- The system shall be aware of proper communication protocol
- The system shall “wait” until radio channel is clear for XX milliseconds prior to communicating via voice
- The system shall communicate reaching cleared altitude
- The system shall communicate reaching decision height
- The system shall communicate reaching waypoint/fix
- The system shall notify human crew when unexpected actions are taken
- The system shall suggest alternate (appropriate) actions to human crew
- The system shall offer to take tasks/actions from human crew
- The system shall increase urgency of alerts and communications when human crew fails to respond within XX
- The system shall communicate taking control when loss of human crew is assumed
- The system shall notify ATC when taking control of aircraft (in the event of loss of human crew)
- The system shall acknowledge and communicate performing action requested by human crew
- The system shall query the human crew if instruction is ambiguous or unexpected/unwarranted

- The system shall query the human crew when unexpected or ambiguous radio call is given
- The system shall query the human crew when unexpected or ambiguous action taken
- The system shall suggest alternate (appropriate) communication responses to human crew
- The system shall review information obtained from ATC against flight plan and electronic data
- The system shall request info from ATC or update electronic data
- The system shall request change/deviation from cleared action if appropriate
- The system shall notify ATC of visual contact with traffic
- The system shall verbally acknowledge location of traffic (and direction moving)
- The system shall read each sequential checklist task and wait for response
- The system shall notify human crew of pertinent weather info and suggest appropriate action
- The system shall monitor pilot control inputs
- The system shall assess control inputs according to SOP, current system state and aircraft orientation
- The system shall query the pilot regarding inappropriate control inputs
- The system shall alert the pilot to control inputs which may compromise safety
- The system shall monitor flight planning and navigation inputs, and
- The system shall inform the pilot of possible errors in data entry
- The system shall inform the pilot of pertinent ATIS and NOTAM information that may impact the planned procedure
- The system shall query the pilot if a more appropriate route is available
- The system shall monitor pilot progress on tasks
- The system shall prompt the pilot if a task is not completed by anticipated completion (time or distance)
- The system shall prompt the pilot of higher priority interrupting tasks
- The system shall remind the pilot of previously initiated tasks when a higher priority task is completed
- The system shall alert the operator if an ATC call is not answered within (time)
- The system shall inform the pilot if the aircraft does not respond within the normal range of expectation in response to control inputs
- The system shall provide real-time self-reporting of current task, progress on task, and tasks in queue
- The system shall effectively communicate current and projected state of automation in terms of ongoing and upcoming task, as well as any issues or errors
- The system shall effectively communicate its current understanding of situation in standard pilot operational terminology