NASA/TM–2015-218988

Overview of Risk Mitigation for Safety-Critical Computer-Based Systems

Wilfredo Torres-Pomales
Langley Research Center, Hampton, Virginia

November 2015

NASA/TM–2015-218988

Overview of Risk Mitigation for Safety-Critical Computer-Based Systems

Wilfredo Torres-Pomales
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

November 2015

Acknowledgment

I would like to express my gratitude to the reviewers and to those who have helped me gain a better appreciation of the complexities in the design and evaluation of dependable systems.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

This report presents a high-level overview of a general strategy to mitigate the risks from threats to safety-critical computer-based systems. In this context, a safety threat is a process or phenomenon that can cause operational safety hazards in the form of computational system failures. This report is intended to provide insight into the safety-risk mitigation problem and the characteristics of potential solutions. The limitations of the general risk mitigation strategy are discussed and some options to overcome these limitations are provided. This work is part of an ongoing effort to enable well-founded assurance of safety-related properties of complex safety-critical computer-based aircraft systems by developing an effective capability to model and reason about the safety implications of system requirements and design.

Table of Contents

Abbreviations v

1. Introduction 1

2. Safety-Risk Mitigation 2
  2.1. Safety-Relevant Functional Characteristics 2
  2.2. Safety-Relevant Quality Attributes 3
    2.2.1. Functional Integrity 4
    2.2.2. Functional Reliability 4
    2.2.3. Functional Recoverability 4
    2.2.4. Functional Availability 4
  2.3. General Safety-Risk Mitigation Strategy 5
  2.4. Conditioning the Fault Space 7
    2.4.1. Prevention of Fault Introduction 7
    2.4.2. Correction of Introduced Faults 8
    2.4.3. Bounding the Fault Scope 8
  2.5. Conditioning the Error Space 9
    2.5.1. Prevention of Fault Activation 9
    2.5.2. Local Containment of Errors 10
    2.5.3. Bounding of Error Conditions 10
  2.6. Architecture-Level Error Mitigation 11
    2.6.1. Fault Hypothesis 11
    2.6.2. Means of Architectural Mitigation 12
    2.6.3. Error Passivation 12
    2.6.4. Error Masking 14
    2.6.5. Error Recovery 15
    2.6.6. Error Mitigation for Single Receiver 16
    2.6.7. Error Mitigation for Multiple Receivers 17
    2.6.8. Error Mitigation for Single Source and Multiple Receivers 19
  2.7. Management of Redundant Groups 23
    2.7.1. Local Structure and Modes 24
    2.7.2. Group Structure and Modes 25
    2.7.3. Dynamic Group Membership 26
    2.7.4. Synchrony 27
  2.8. Safety-Risk Mitigation Limitations 27
  2.9. Robust Resilience 28
  2.10. Other Sources in the Literature 29

3. Final Remarks 31

References 32

Abbreviations

A_A	Acceptable-Item Asymmetry
AA_m	Failed Active Asymmetric
A_m	User Asymmetry
AS_m	Failed Active Symmetric
BIST	Built-In Self-Test
C	Correct
CCA	Common Cause Analysis
CMA	Common Mode Analysis
ConOps	Concept of Operations
COTS	Commercial Off The Shelf
CS	Correct Symmetric
D	Detectable
DIMA	Distributed Integrated Modular Architecture
DMA	Direct Memory Access
DPS	Data Processing System
ECR	Error Containment Region
ECXF	Error Containment Interface Function
ESD	Electrostatic Discharge
FAA	Federal Aviation Administration
FCR	Fault Containment Region
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GOI	Group of Interest
IEEE	Institute of Electrical and Electronics Engineers
IFR	Internal Failure Recovery
IKIWISI	I'll-know-it-when-I-see-it
IMA	Integrated Modular Avionics
IoP	Index of Performance
IUE	Initiating Unintended Event
LoF	Loss of Function
LRU	Line Replaceable Unit
MF	Malfunction
MIL-STD	Military Standard
NAS	National Airspace System
NASA	National Aeronautics and Space Administration
ORS	Operation Requirements Satisfied
ORV	Operating Requirements Violated
OS	Omissive Symmetric
OS_m	Operational Symmetric
OTH	Omissive-Transmissive Hybrid
PA_m	Failed Passive Asymmetric
PRA	Particular Risk Analysis
PS_m	Failed Passive Symmetric
RAF	Recoverable Active Failure
RCV	Receiver
RF	Radio Frequency
RPF	Recoverable Passive Failure
RRS	Recovery Requirements Satisfied
RRV	Recovery Requirements Violated

S_A	Acceptable-Item Symmetry
SDOA	Single-Data Omissive Asymmetric
S_m	User Symmetry
SOA	Strictly Omissive Asymmetric
SOI	System Of Interest
SRC	Source
SS	Sub-System
TA	Transmissive Asymmetric
TRL	Technology Readiness Level
TS	Transmissive Symmetric
TTC	Time To Criticality
TTE	Time To Effect
TUE	Terminal Unintended Event
U	Undetectable
UAF	Unrecoverable Active Failure
UPF	Unrecoverable Passive Failure
V&V	Validation and Verification
ZSA	Zonal Safety Analysis

1. Introduction

An aircraft consists of a collection of systems performing a wide variety of functions with different levels of safety-criticality. The aviation industry is continuing a decades-old trend of adopting increasingly sophisticated computer-based technology to implement aircraft functionality. Modern aircraft are highly complex, functionally integrated, network-centric systems of systems [1]. The design and analysis of distributed-computation aircraft systems are inherently complex activities. Ensuring that such systems are safe and comply with existing airworthiness regulations is costly and time-consuming as the level of rigor in the development process, especially the validation and verification activities, is determined by considerations of system complexity and safety criticality. A significant degree of care along with deep insight into the operational principles of these systems are necessary to ensure adequate coverage of all design implications relevant to system safety.

Validation and verification (V&V), as well as certification, of complex computer-based systems, including safety-critical systems, are recognized problems of national significance [2], [3], [4]. The challenges in assuring the design and safety of complex systems require considerable attention and financial investment [5]. As aircraft system complexity continues to increase, V&V and certification costs, together with related programmatic risks, can provide a basis against the development and implementation of new capabilities [6]. Such obstacles against innovation pose a threat to national competitiveness and can hinder the proposed operational improvements to the National Airspace System (NAS) that are intended to increase capacity and flexibility as well as reduce costs, but would also increase the complexity of airborne and ground aviation systems [7]. There are initiatives underway to produce methods, tools, and techniques that enable predictable, timely, and cost-effective complex systems development [8]. NASA aims to identify technical risks and to provide knowledge to safely manage the increasing complexity in the design and operation of vehicles in the air transportation system. In furtherance of this goal, multidisciplinary tools and techniques are being developed to assess and ensure safety in complex aviation systems and enable needed improvements to the NAS.

This document is a contribution to a design and evaluation guide currently under development. The guide is intended to (1) provide insight into the system safety domain, (2) present a general technical foundation for designers and evaluators of safety-critical systems, and (3) serve as a reference for designers to formulate well-reasoned safety-related claims and arguments and identify evidence that can substantiate these claims. This evidence forms a basis for demonstrating compliance with certification regulations. The generation of such evidence is a major objective of a system development process. This report is part of an ongoing effort to enable justifiable assurance of safety-related properties of computer-based aircraft systems by developing an effective capability to model and reason about the safety implications of system requirements and design.

This document presents a general strategy to mitigate the risks from threats to safety-critical computer-based systems. In this context, a **threat** is a process or phenomenon that can cause operational safety hazards in the form of computational system failures. The safety-risk mitigation strategy is intended to achieve a desired level of system dependability measured in terms of qualities based on essential characteristics of failure causes and effects. The presentation is a high-level overview intended to provide insight into the safety-risk mitigation problem and potential solutions. The limitations of the strategy are discussed and some options to overcome these limitations are provided. The document also serves as an introduction to the extensive body of knowledge on safety-critical computer-based systems.

2. Safety-Risk Mitigation

The mitigation of risks from system safety threats requires consideration of likelihood (or frequency), severity, and uncertainty of the threats and their effects. In this report, it is assumed that the system functional and performance requirements are safe in the sense that a compliant system service will not cause a mishap. The risk mitigation goal is to ensure that the system is **dependable** in the sense that the residual risk level of operational service failures is acceptable to the stakeholders, including users, regulatory authorities, and developers, among others. At a high level, two relations determine risk: the relation between likelihood and severity of system failures, and the relation between the degree of uncertainty and the severity of system failures. There are two kinds of uncertainties: **aleatoric** uncertainty due to inherent randomness or variation; and **epistemic** uncertainty due to modeling abstractions and lack of knowledge. These uncertainties are about the threats, the system itself, and the environment. Generally, for safety-critical systems, both of the risk relations are inverse relations, each with a level no higher than a given maximum threshold for acceptable risk. In effect, as the potential severity of failure scenarios increases, we want the failure frequency to decrease and remain below the maximum acceptable level, and we also want to have decreasing uncertainty (i.e., increasing confidence) about the failure frequency and severity estimates.

This section provides insight into the means to mitigate safety threats. It begins with a review of the functional and quality attributes of a dependable system. A general safety-risk mitigation strategy is introduced which identifies the requirements and opportunities to effect mitigation of non-operational and operational threats. This section also describes generic architectural structures for threat mitigation during system operation. Various aspects of the problem of group redundancy management are examined, including options for system structures and component interaction protocols based on the types of faults that are expected in a system. In addition, this section describes the limitations of architectural techniques from the perspective of fault tolerance guarantees. Finally, this section provides a complementary perspective of best-effort robustness and recovery constrained by available resources and other system conditions.

2.1. Safety-Relevant Functional Characteristics

A basic high-level functional safety assessment with a failure model defines three possible static functional states: operational (i.e., not failed), failed passive, and failed active. A **passive failure** state is a loss-of-function condition in which the function is not being performed. An **active failure** corresponds to a malfunction in which the function is performed incorrectly. Passive and active failures are also known as **omission** and **commission** failures, respectively.

A primary goal of worst-case functional safety assessments is the definition of bounds on the severity of failure modes and effects. We want to determine the conditions necessary to guarantee particular functional failure modes and effects. To this end, instead of assuming static functional states as above, we can define the state of the system under an increasingly permissive behavioral classification hierarchy in which a function can be in one of three possible states:

- **Operational**, if only proper functional service is being delivered;
- **Failed passive**, if the functional service is a combination of operational and passive failure; and

- **Failed active**, if the functional service is a combination of operational, passive failure, and active failure.

An operational functional service is a subset of a failed-passive functional service, which is a subset of a failed-active. Notice that in this model a failed-active service corresponds to an **arbitrary** failure with no constraints on the behavior exhibited by the system. Intuitively, we want a safety-critical system to either operate properly or not operate at all (i.e., stop), rather than operate in an arbitrary manner. The determination of the system state is based on the level of behavioral constraint that can be guaranteed at a particular point in time.

In a **real-time** (i.e., **time-critical**) system service, the correctness of the service is determined not only by the value of the service items, but also the time of delivery [9]. For real-time functions, the severity of a failure may depend on the duration of the failure condition. A **hard real-time** service must always deliver service items within the specified time interval, as there may be highly undesirable consequences to the users if this constraint is violated. A **soft real-time** service may fail to deliver service items within the specified time constraint, but the utility of the item decreases when the constraint is violated [10]. Some systems have **firm real-time** service requirements in which infrequent timing constraint violations are tolerable but may degrade the quality of the service. Some systems may be firm real-time with respect to the quality of the service, but hard real-time with respect to safety. For these systems, the quality of the service degrades as the update delay increases beyond the firm timing constraint until the hard real-time constraint is reached, at which point safety is compromised. This hard real-time delay threshold corresponds to the **time-to-criticality** of a system, which is the time interval between the occurrence of a failure and the user or environment reaching an unsafe state. For highly dynamic functions, the time to criticality can be very short and failure recovery within that time may be unfeasible or require an automated capability. For example, Paulitsch et al. [11] and Pimentel [12] reference a design requirement of 50 ms maximum service outage duration for an automobile steer-by-wire system. For less dynamic functions, automatic recovery may be possible, with even manual recovery by a human operator being adequate.

It is assumed that the system of interest (SOI) is embedded in a larger technical or socio-technical operations system (i.e., a system consisting of people and technology) that is intended to achieve higher-level goals. Consequently, safety in the operations system depends on the characteristics of the function performed by the SOI. The acceptable level of risk for the dependence of the operations system on the SOI is determined by the strength (i.e., the importance or criticality) of the relation between safety in the operations system and the performance of the SOI. The **frequency** of SOI failures is determined by the duration of continuous operation between failures and the time to restore service after experiencing a failure. For a real-time SOI whose service correctness criteria includes timing constraints, the **severity** of SOI failures is determined by the failure mode of the SOI and the duration of the failure condition. The SOI service characteristics of **time-to-failure**, **time-to-recover**, and **failure mode** depend on the characteristics of the threats and the architecture of the SOI. In general, the system architecture is designed to achieve the desired mapping of characteristics between threats and service quality by increasing the time-to-failure, reducing the time-to-recover, and reducing the complexity and severity of the failure modes (i.e., maximizing the desirable attributes and minimizing the undesirable ones).

2.2. Safety-Relevant Quality Attributes

As the required SOI function is presumed to be safe, we are interested in safety-critical qualities of the

SOI service. Based on the functional characteristics described in the preceding sub-section (i.e., time-to-failure, time-to-recover, and failure mode), the dependability of the SOI is determined by the timing of operational and failure conditions and the characteristics of the failure modes. It is important to notice that dependability and safety are determined, not only by the SOI failure modes, but also by the timing of failures and recovery. The desired service dependability can be specified in terms of the qualities of functional integrity, availability, reliability, and recoverability.

2.2.1. Functional Integrity

In a general sense, integrity is related to failure modes of a system and the concepts of truthfulness and trustworthiness. Avizienis et al. [13] defined integrity as the absence of improper system state alterations. Paulitsch et al. [11] defined integrity as the probability of an undetected failure. Functional integrity is an important functional quality related to the potential that the effects of an SOI failure will propagate and corrupt the operations system (i.e., the larger system that encompasses the SOI). The SOI satisfies this condition when it is operational or failed passive. Integrity is violated when the system is failed active. **Functional integrity** can be measured as the probability that the SOI will not experience an active failure during a specified time interval under stated conditions. These stated conditions can be physical environmental conditions (e.g., temperature, vibration, etc.), the configuration of the SOI, the functional input patterns, and possibly the types and number of faults experienced by the SOI.

2.2.2. Functional Reliability

Reliability refers to the uninterrupted delivery of correct service [13]. **Reliability** is measured as the probability that the SOI function will remain operational for a specified time interval under stated conditions [9]. Reliability determines the time-to-failure characteristic of the SOI.

2.2.3. Functional Recoverability

Recoverability is the ability to restore service delivery after experiencing a failure. Here we use the term recoverability to refer to the ability of a system to restore service on the fly. This falls under the larger context of maintainability, which includes physical replacement and repair of system components. For our purpose, recoverability is the complement of reliability and is measured as the probability that the service is restored within a specified time interval under stated conditions. Recoverability determines the time-to-recover characteristic of the SOI.

2.2.4. Functional Availability

Availability refers to the fraction of time that the delivered service is correct. **Availability** is measured as the probability that the SOI is operational during a specified time interval under stated conditions. Availability is a function of reliability and recoverability, and it combines the characteristics of time-to-failure and time-to-recover. Availability is highest when both reliability and recoverability are high, as in this case the SOI remains operational for long time intervals and quickly recovers after experiencing a failure.

2.3. General Safety-Risk Mitigation Strategy

The functional safety goal is for the SOI to be dependable in the sense that the frequency and severity of functional failures are not higher than is acceptable. This is accomplished by applying the **defense-in-depth** concept (i.e., multiple layers of protection) with non-operational and operational means to prevent and mitigate system failures. This approach mitigates the risk of failures by reducing the likelihood of the causes and the severity of the effects.

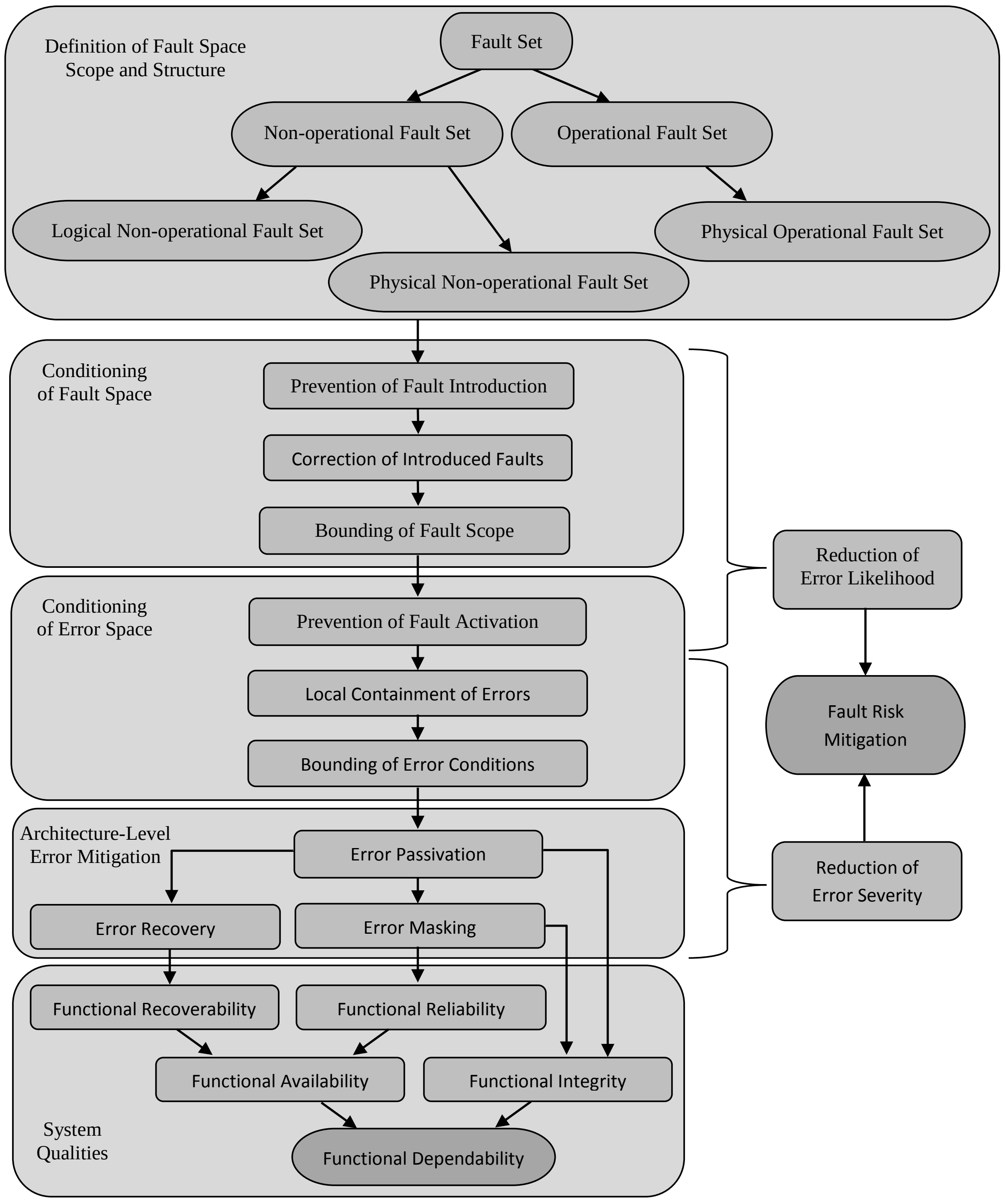
Figure 1 illustrates a general strategy to mitigate the risk of system-safety threats and the faults (i.e., defects) they can introduce in a system. The strategy is applicable to the full set of possible faults, including physical and logical faults of non-operational and operational nature. This approach leverages the causal chains in a threats-and-effects model. With this model, threats to the SOI cause faults that remain latent in the system until there are favorable conditions for their activation in the form of errors. Once activated, errors may propagate until they eventually cause service failure at the external interfaces. The major layers of defense are:

- Conditioning the fault space,
- Conditioning the error space, and
- Architecture-level error mitigation.

Multiple layers of defense are needed to support each other because no single layer can ensure complete containment of fault causes and their effects. In general, there is a residual uncertainty about the effectiveness of individual layers of defense and about the relationship between the weaknesses of different layers of defense.

The first major layer of defense is conditioning the fault space in the physical and logical layers of the SOI. In this context, conditioning refers to creating favorable conditions in the fault space applicable to the SOI. This is achieved by minimizing the number of faults present in the system and the scope (i.e., extent) of each fault. The number of faults is minimized by preventing the introduction of non-operational and operational faults and by correcting faults that are discovered. The scope of the remaining faults is minimized by ensuring adequate independence in the introduction of primary faults and between primary and secondary faults. This is accomplished by minimizing the strength of causal couplings between primary faults that are directly caused by threats (i.e., minimize the likelihood that a single threat event introduces multiple faults), and by minimizing the causal couplings between primary and secondary faults due to cascade propagation effects.

As shown in Figure 1, the second major layer of defense is conditioning the error space in the information layer. This can be accomplished by preventing the activation of faults, for example, by workarounds that circumvent known existing faults. Per Figure 1, the next way to manage the error space is to provide mechanisms in the SOI components to contain fault-induced errors by locally detecting them and, if possible, correcting them. If faults become active and their effects cannot be contained locally, we want to ensure that the scope of the error conditions does not overwhelm the protection mechanisms at the architecture level.



Definition of Fault Space Scope and Structure
Fault Set
Non-operational Fault Set
Operational Fault Set
Logical Non-operational Fault Set
Physical Operational Fault Set
Physical Non-operational Fault Set
Conditioning of Fault Space
Prevention of Fault Introduction
Correction of Introduced Faults
Bounding of Fault Scope
Reduction of Error Likelihood
Conditioning of Error Space
Prevention of Fault Activation
Local Containment of Errors
Bounding of Error Conditions
Fault Risk Mitigation
Architecture-Level Error Mitigation
Error Passivation
Error Recovery
Error Masking
Reduction of Error Severity
Functional Recoverability
Functional Reliability
Functional Availability
Functional Integrity
System Qualities
Functional Dependability

Figure 1: General Strategy for Safety-Risk Mitigation

The third major layer of defense is the mitigation of errors by the architecture of the SOI. The fundamental requirement is to contain the propagation of errors in the system. If containment is accomplished, the system can mask the errors to prevent the external service from being affected in any way, or it can recover from errors by correcting the service after a failure.

The layers of defense in this strategy combine to mitigate the overall risk posed by the threats. The conditioning of the fault and error spaces is intended to reduce the likelihood of error conditions in the SOI, especially complex error conditions that could overwhelm the fault handling capabilities of the architecture. The architecture-level error mitigation is intended to reduce the severity of error conditions by constraining the system failure modes and minimizing their duration. The combined architecture-level mitigations of passivation (i.e., to make passive), masking, and recovery deliver the desired functional qualities of integrity and availability, and together result in a dependable system with an acceptable risk of failure. As stated previously, there is always uncertainty about the effectiveness of the employed means of risk mitigation. Any assessment of a risk mitigation approach against system failures must consider the three risk components of likelihood, severity, and uncertainty, which are dependent on random variation and knowledge of the threats, the system, and the environment.

The following sections provide insight into this risk mitigation strategy, including implications and limitations.

2.4. Conditioning the Fault Space

The motivation for conditioning the fault space is to have favorable conditions for dependable system operation. The goal is to minimize the number and scope of faults present in the SOI. These faults can be non-operational or operational in origin and can be in the physical or logical layers of the system. This is accomplished by preventing the introduction of faults, correcting discovered faults, and maximizing the independence of introduced faults.

2.4.1. Prevention of Fault Introduction

This layer of defense targets the introduction of physical and logical faults in all phases of the system life cycle. Non-operational faults can be introduced during development, production, manufacturing, refinement, and maintenance of the system. Guidelines, standards, policies, and regulations for development and other phases of the system life cycle have been created to ensure a minimum adequate level of rigor and quality. These life cycle measures aim to achieve a higher level of quality for the most critical components relative to the desired system properties. Some of the system development standards in the aviation industry include SAE International Aerospace Recommended Practice ARP-4754A Guidelines for Development of Civil Aircraft and Systems [14], RTCA DO-254 Design Assurance Guidance for Airborne Electronic Hardware [15], and RTCA DO-178 Software Considerations in Airborne Systems and Equipment Certification [16]. There are many other sources of information on recommended and standard practice for system life cycle quality assurance. In general, the level of effort and rigor to ensure the quality of a system is determined by its complexity and criticality. Operational faults can be prevented by the selection of electrical (e.g., diodes and integrated circuits) and mechanical (e.g., enclosures) system components with an adequate level of quality, and by ensuring that the system is able to tolerate the stresses in the expected physical environment, such as specified in the standard RTCA DO-160 Environmental Conditions and Test Procedures for Airborne Equipment [17]. Equally important is ensuring that the system is not exposed to harsher environmental conditions than that for which it is qualified.

In spite of these measures, it generally cannot be guaranteed that faults are not introduced into a system. The best that can be expected is a minimization of the likelihood of introducing non-operational and operational faults.

2.4.2. Correction of Introduced Faults

The next layer of defense is to correct faults after they are introduced. This is applicable to non-operational and operational faults in the physical and logical layers of the system. V&V activities examine the system to determine whether the requirements are correct and complete according to the intended system purpose, and also to determine whether the implementation complies with the requirements. The types of V&V activities include tests, analyses, and reviews of the system and its life cycle data. In addition to non-operational V&V activities, faults may also be discovered during operation if the system or its components fail to perform their intended function. In that case, offline verification and analysis could be used to diagnose the failure and identify the causal defect(s). This applies to system development and refinement faults as well as faults introduced during manufacturing, production, and maintenance processes.

2.4.3. Bounding the Fault Scope

If we cannot prevent the introduction of faults or correct them all, then we need to ensure that the faults present in the system are causally unrelated or minimally related. The prime motivator for this strategy is the presumed correlation between shared causality of faults and similarity of manifestations in value and time. Another way of stating this heuristic principle is that *things that are different and unrelated fail differently and at different times*. The minimization of causal relations must address the introduction of multiple primary faults with common causal threats and multiple secondary faults with common causal primary faults. This minimization of causal relations effectively bounds the scope of introduced faults.

Common Cause Analysis (CCA) as described in SAE International Aerospace Recommended Practice ARP-4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment [18] is intended to ensure that faults in the system are independent or that the risk associated with dependence is acceptable. CCA is divided into three types of analyses: Zonal Safety, Particular Risk, and Common Mode.

Zonal Safety Analysis (ZSA) is a qualitative analysis to verify independence claims for component failures. ZSA examines the effects of component failures on other components in physical proximity and the implications of errors during maintenance actions. ZSA is performed for each zone of a vehicle during development and modifications to the vehicle.

Particular Risk Analysis (PRA) examines particular events or conditions external to the system that have the potential to cause violation of independence claims. Examples of these threats include fire, fluids, lightning, high intensity radiated electromagnetic fields, and break-up and explosion of mechanical systems. The PRA aims to identify direct and cascade effects of these threats in order to eliminate or mitigate the safety risk.

Common Mode Analysis (CMA) is a qualitative analysis to verify independence claims related to design, manufacturing, and maintenance errors. The analysis is intended to provide the technical basis to eliminate or minimize common design, manufacturing, and maintenance conditions that could violate independence claims. This analysis covers aspects such as hardware faults, software faults, production and repair flaws, environmental factors, requirement errors, cascading effects, and external sources of faults. A CMA examines the system for commonalities for each of the lifecycle stages and operational conditions such as concept and design (e.g., architecture, technology, and specifications), manufacturing (e.g., manufacturer and procedures), installation (e.g., location and routing), operation (e.g., staff and procedures), and environment (e.g., temperature, vibration, humidity, particle radiation, and electromagnetic radiation).

An approach to bound the scope of faults is to define and enforce confinement boundaries that prevent the indiscriminate propagation (i.e., a cascade) of faults throughout a system and the generation of secondary faults. A **fault containment region** (FCR) is carefully crafted with fault isolation mechanisms and common-cause analyses, as described above, to ensure containment of propagation into and out of each FCR. An FCR is effectively a discrete unit of failure in a system. Ideally, faults introduced in different FCRs are different and unrelated. However, some threats (e.g., lightning and design errors) may be able to influence multiple FCRs, thus compromising failure independence. The goal is then to achieve an adequate level of containment in order to bound the influence of introduced faults to their respective FCRs. Because different threats may have different scopes of influence in the introduction of faults, it may be possible and advantageous to structure a system with a hierarchy of FCRs that provide multiple layers of containment, and that promote containment within a region no larger than the scope of direct influence of threats relative to the introduction of primary faults (i.e., to prevent fault propagation and generation of secondary faults in multiple FCRs).

None of the layers and existing techniques for conditioning the fault space offers a guarantee of effectiveness. Furthermore, the effectiveness of the composition of layers of protection is not guaranteed either. The available approaches are mostly heuristic and qualitative and their effectiveness is assessed based on engineering judgment and compliance with best practices. Additional layers of protection may be needed to reduce the risk to an acceptable level.

2.5. Conditioning the Error Space

The next major layer of defense is conditioning errors in the SOI information layer that may be generated by faults present in the system. The approaches and techniques for error conditioning must account for non-operational and operational faults in the physical and logical layers of the SOI. The overall goal is to minimize the number and scope of error conditions that must be handled by architecture-level fault-tolerance mechanisms in the system.

2.5.1. Prevention of Fault Activation

If there are known faults present in the system, it may be possible to manage the external and internal operation of the SOI to prevent the generation of errors. For example, if certain input patterns to a component cause it to generate errors, there may exist an alternative input sequence (in effect, a work-around) with a different but equivalent input pattern that allows the component to compute a correct output without the activation of internal faults. This could be achieved by a re-expression or a decomposition and reordering of the input sequence [19]. Another approach to prevent the activation of known faults in a particular component is to switch to an alternate component that provides an identical or equivalent function. Also, a component or the whole system may have reversionary (i.e., alternate) modes of operation that reconfigure the internal data flow to deliver alternate and possibly simpler but effective functionality. The flight control systems in Airbus airplanes [20] [21] [22] and the Boeing 777 airplane [23] are good example of systems with reversionary modes. Notice that fault activation prevention is a general approach that can be applied to manage both physical and logical faults.

2.5.2. Local Containment of Errors

This layer of defense exploits the bounded scope of faults, the hierarchical structure of FCRs within system components, and redundancy within a component for self-checking its internal operation. A component may perform both offline and online checks to detect internal faults and locally contain the propagation of errors. Some of the possible kinds of checks include timing checks (e.g., processing delays), coding checks (e.g., checksums), reasonableness of computation results based on known semantic properties, and checks based on properties of data structures [19]. Hardware circuits may have built-in self-test (BIST) and self-checking logic [24]. Periodic background scrubbing of data protected by error control codes in memory and programmable logic circuits is another technique to prevent the propagation of local errors [25]. All these techniques are effective and may have very high coverage for certain types of errors. For some applications, these local error checks may be adequate to achieve an acceptable level of error containment. However, in general, this sort of self-checking cannot guarantee complete coverage for all possible or relevant faults and error manifestations.

2.5.3. Bounding of Error Conditions

At this point, the goal is to do whatever is reasonable to minimize the size and complexity of error conditions that must be handled by the system architecture. Ideally, the system components fail independently and with easy-to-handle failure modes. Simpler failure modes can be mitigated with fewer resources, thus allowing the architecture to remain relatively simple and focused primarily on optimal implementation of the system function. Component failure independence is intended to ensure that the failures are scattered spatially and temporally, allowing the architecture to deal with them locally and one at a time. This means that the time between error arrivals is larger than the time to recover from them. We also want to minimize the likelihood of coincident component failures, even if they are relatively simple. One concern for time-coincident (i.e., concurrent or simultaneous) component failures is the possibility of coupling or correlation of the failure modes such that the components effectively collude to create conditions that are much more difficult to mitigate.

This is the last layer of defense before architecture-level error mitigation. We would like the error space at this point to be characterized by an inverse relation between the likelihood and severity of error conditions. This means that the architecture would most often have to handle relatively simple failure modes, while still having the means to protect against uncommon complex error conditions. Because of the importance of achieving this error risk relation, there should also be an inverse relation between the degree of uncertainty about the error conditions and their level of severity. The characteristics of the error space achieved at this point in the safety-risk mitigation strategy will be a primary factor in the definition of the system fault hypothesis (i.e., fault assumptions), which is the design basis for architecture-level error mitigation.

2.6. Architecture-Level Error Mitigation

Error mitigation at the architecture level is the third layer of defense in the proposed safety-risk mitigation strategy. Architecture-level mitigation provides resources and mechanisms to achieve an acceptable level of risk due to errors originating within the SOI.

The SOI is required to have a certain level of functional quality for a specified mission duration and stated operational conditions. The principal quality of interest for the systems considered in this report is dependability. As described previously, dependability is the maximum acceptable risk for the SOI function and it is specified in terms of the attributes of functional integrity and availability (including reliability and recoverability). The exposure of a particular function during a mission (i.e., the amount of time that there is a need and demand for the function) can range from minutes up to full mission duration, depending on the kind of function and mission phases. The operational conditions include the specification of the threats that the SOI may face during a mission.

The architecture of the SOI provides the link between the assumptions (i.e., the stated operational conditions) and the required system properties (i.e., the system guarantees). These assumptions and guarantees cover both function and quality aspects of the system, including dependability. The architecture of the SOI specifies the composition of the system in terms of components (i.e., sensing, computation, and actuation nodes), their communication interconnections, and the interaction protocols. Each of these architectural elements has its own assumptions and guarantees for function and quality. The dependability of the system is a function of the dependability of the architectural elements.

2.6.1. Fault Hypothesis

The fault hypothesis (or assumption) is the effective threat risk the SOI architecture must mitigate to achieve the required functional dependability during a mission. These are the relevant fault and error conditions in the physical, logical, and information layers of the internal components, interconnections, and interaction protocols. These fault and error conditions are the result of the combined effect of the threats to the SOI and the mitigation provided by the layers of defense of fault space and error space conditioning. Henceforth, the fault hypothesis of the SOI will be referred as the **architecture-level threat** (AT) **hypothesis**.

In general, the SOI must contend with two major types of faults and errors: exogenous and endogenous. Exogenous faults and errors are not relevant to the dependability of the SOI itself. In this report, it is assumed that the SOI as a whole forms a fault-containment region in the sense that faults external to the SOI do not cause secondary faults within the SOI, and faults originating within the SOI do not propagate outside its boundary. In addition, external errors are not considered in the dependability of the SOI. Only faults internal to the SOI and their effects are relevant to the dependability of the SOI. In effect, in determining the dependability of the SOI, it is assumed that the environment has perfect dependability (i.e., no failures). Note that in a context of a larger system containing the SOI, errors at the inputs to the SOI may propagate and influence the observed dependability at the outputs of the SOI. Such a scenario would be relevant to the dependability of the containing system.

The AT hypothesis consists of two major parts: fault containment regions (FCRs) and dependabilities of architectural elements (i.e., components, interconnections, and interaction protocols). The FCRs are subsets of architectural elements assumed to experience physical and logical defects with a high degree of probabilistic independence from other subsets [26] [27] [28] [29] [30] [31]. This is achieved by minimizing the causal couplings among primary faults and between primary and secondary faults (see preceding section on Conditioning the Fault Space). The dependabilities of architectural elements describe their failure frequencies and modes relative to their own faults. The failure frequencies of architectural elements are related to their time-to-failure and time-to-recover characteristics. The dependabilities of architectural elements can be specified in terms of their own functional integrity and availability (including reliability and recoverability).

The AT hypothesis can be stated in probabilistic or deterministic terms. A probabilistic AT hypothesis would include the probability of coincident FCR failures and the functional integrity and availability stated in terms of failure occurrence rates (e.g., 10^{-6} per hour) or probabilities for a given mission duration. A deterministic AT hypothesis describes the scenarios that the system may encounter, such as number of sequential or simultaneous internal failures, the failure modes, and failure durations.

2.6.2. Means of Architectural Mitigation

A safety-critical computer-based system has two major aspects: function and safety. The safety aspect complements the function in order to achieve the required level of system dependability. The system safety features are necessary for performing the required system function only because of the risk due to non-operational and operational faults. These safety features are introduced for handling error conditions at the boundary and within the system, including failures of the safety features themselves. The system architecture defines the error mitigation strategy, including the extent of the safety features and the level of integration between the function and safety features. Taken together, the system can be viewed as a set of integrated functional and safety resources and mechanisms with a management policy designed to realize the system function with the required dependability.

As indicated in the general safety-risk mitigation strategy (see Figure 1), system functional dependability can be specified in terms of functional integrity and functional availability. Functional availability can be decomposed into functional reliability and functional recoverability. An architecture-level error mitigation strategy achieves system dependability by enforcing constrained system-level failure modes and by enabling the system to either remain operational or fail and recover quickly after internal failures. The means of architectural mitigation are error passivation, masking, and recovery. As shown in Figure 1, functional integrity and functional reliability depend on error passivation and masking. Error passivation and recovery are needed for functional recoverability.

The architecture-level error mitigation capabilities of passivation, masking, and recovery are realized by local and global error handling mechanisms. A fundamental principle of an error mitigation strategy is the granularization of failure by defining and enforcing **error containment regions** (ECRs). ECRs complement FCRs in the physical and logical layers to define independent units of failure. ECRs can be structured hierarchically to provide multiple layers of containment between the points of origin of internal failures and SOI outputs. Local ECR boundary (i.e., interface) enforcement is achieved by leveraging good operating resources to detect and correct errors from failed resources thus achieving the desired ECR functional dependability. Local ECR dependability is complemented with error-mitigating global interaction protocols that ensure desired system properties in the presence of a bounded number of failed participating ECRs.

2.6.3. Error Passivation

As asserted earlier, from a safety perspective, a function has three possible states: operational (i.e., not failed), failed passive (i.e., failure constrained to a loss of function), and failed active (i.e., unconstrained malfunction). This classification is relative to the worst-case guarantee that can be offered at a point time, which may not be the same as the actual state of the function. Therefore, a failed passive function may be operational or failed passive, and a failed active function is essentially arbitrary. In general, unconstrained active failures and their associated effects are inherently unsafe and undesirable conditions because the consequences are unknown and possibly unbounded, including the possibility of cascade (i.e., uncontained) secondary failures that violate failure independence assumptions.

Error passivation is the ability to constrain and contain active failure effects. The goal in error passivation is to prevent active failure effects from propagating onto a good FCR and corrupting its state, effectively rendering it failed and potentially becoming a secondary source of active failure effects. At a conceptual level, error passivation has two basic elements: self-checking and self-protection [32]. These are illustrated in Figure 2 and Figure 3, respectively. In these figures, SRC and RCV denote source and receiver, respectively. In **self-checking** error passivation, the output of a source-function FCR is checked by an independent source-safety FCR before being sent out to a receiver FCR. In this configuration, the source-function FCR and source-safety FCR form an ECR. In **self-protection** error passivation, the input from a source FCR is checked by an independent receiver-safety FCR before forwarding it to the receiver FCR. Here the receiver-safety FCR and receiver FCR form an ECR. In the self-checking and self-protection configurations, the safety FCRs are solely responsible for error passivation. The effectiveness of these configurations is limited by the error coverage of the inline safety checks. For some applications, inline syntactic and semantic error checks may be adequate to achieve the required level of error passivation. However, Meyer and Sundstrom have shown that the probability of fault detection for a component can be made equal to unity only if the detector is as complex, in terms of number of states, as the component being monitored [33]. Thus, the only way to achieve the extremely high probability of error passivation required for critical applications is to use independent redundant copies of FCRs. This is illustrated in Figure 4 and Figure 5, where there are now two source FCRs and the safety check is a comparison of delivered services performed at the source ECR or at the receiver ECR, respectively. The optimal error passivation configuration for a particular application may be a combination of source-side and receiver-side checks with or without redundancy.

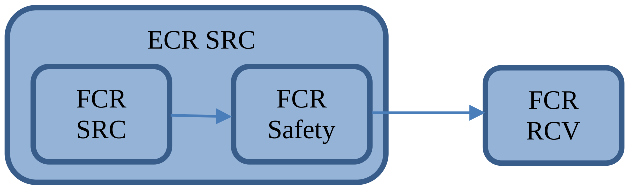


Figure 2: Self-Checking (Source-based) Error Passivation Concept

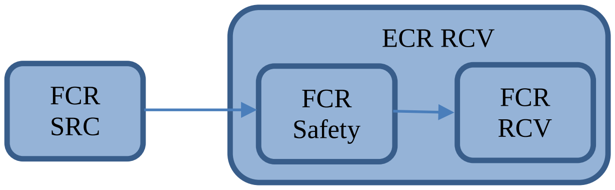


Figure 3: Self-Protection (Receiver-based) Error Passivation Concept

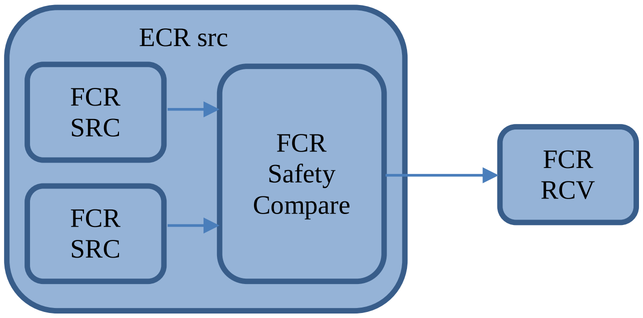


Figure 4: Self-Checking Error Passivation with Source-side Redundancy Comparison Check

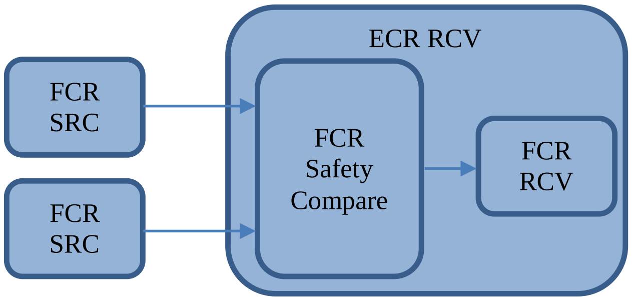


Figure 5: Self-Protection Error Passivation with Receiver-side Redundancy Comparison Check

2.6.4. Error Masking

Error masking is the ability to preserve the operational state of a function. The threat assumed here is a passive FCR failure, and the goal is to ensure that the delivered function effectively remains in the operational state after the failure of the performing FCR. This is an error handling capability in which functional failure effects are contained and corrected without disrupting the delivered service. Self-checking and self-protection ECR structures as illustrated in Figure 6 and Figure 7 can be applied to realize this capability. The selection function in the FCR Safety Select block outputs the first valid input (i.e., an input that is not detectably incorrect). Both configurations use independent redundant source FCRs to ensure that at least one is operational and the safety FCR can forward the output of an operational source FCR without interruption.

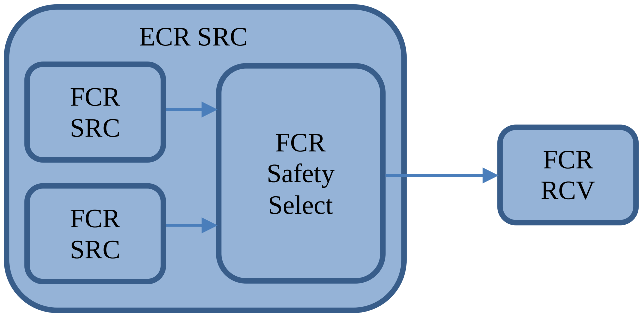


Figure 6: Self-Checking (Source-based) Error Masking

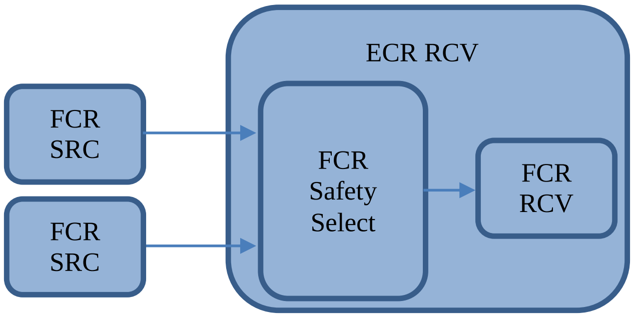


Figure 7: Self-Protection (Receiver-based) Error Masking

2.6.5. Error Recovery

Error recovery is the ability to restore a function to operational state. The assumed threat is a passive FCR failure. This capability ensures that service will be restored if required resources are available, but uninterrupted service is not guaranteed. Self-checking and self-protection configurations are possible to realize an error recovery capability, as illustrated in Figure 8 and Figure 9. The source FCR performs the function of interest, and the safety FCR is responsible for signaling a failure condition, which triggers a recovery action such as restarting the source FCR or enabling an alternate source FCR to re-establish operational functional state as illustrated in Figure 10. The safety FCR can be contained in a source ECR or a receiver ECR.

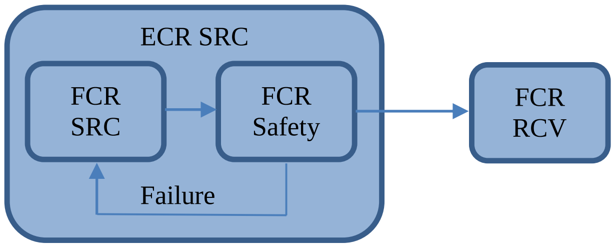


Figure 8: Self-Checking (Source-based) Error Recovery

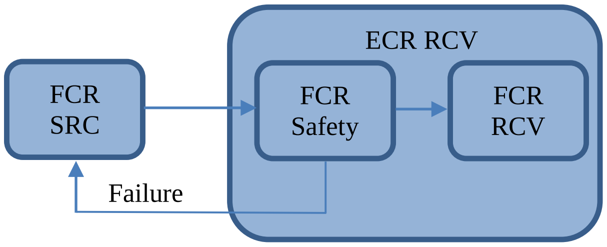


Figure 9: Self-Protection (Receiver-based) Error Recovery

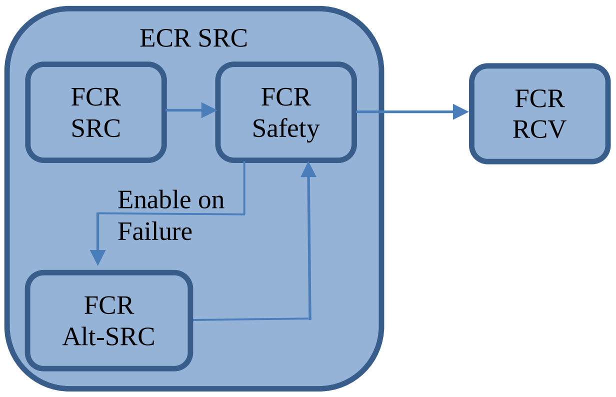


Figure 10: Self-Checking Error Recovery with Alternate Source

2.6.6. Error Mitigation for Single Receiver

The preceding mitigation capabilities can be combined to achieve the desired degree of error mitigation at a receiver for a given failure mode of the source. This configuration is illustrated in Figure 11, where there are up to *n* sources and the receiver has an error containment interface function (ECXF) to manage the inputs. Table 1 shows the required minimum number of sources *n* assuming the number of failed sources is at most *F*. The Table also lists the ECXF function for a given worst-case source failure mode as well as the desired effect at the output of the ECXF. These configuration parameters are determined by assuming that the outputs of operational sources agree (i.e., they have either exact or approximate equality). The **vote** interface function can be a majority, mid-value, or mid-point select, depending on the context. Of note is the general approach of using good operational sources to mitigate the effects of failed sources. Also, note that the desired effect of this approach is to propagate to the receiver the agreed-upon output of operational sources, or to fail in a passive way. An operational worst-case effect supports both functional availability and integrity requirements at the system level, and the passive worst-case effect supports the functional integrity requirement.

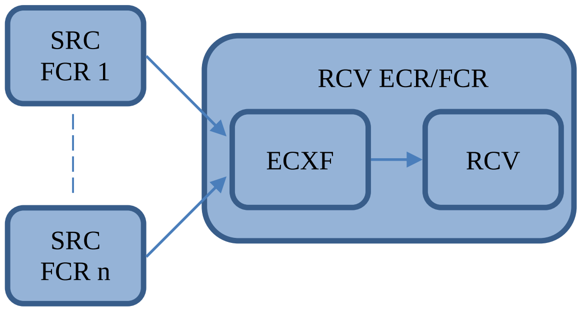


Figure 11: Error Mitigation Configuration for Single Receiver

Table 1: Error Mitigation Parameters for Single Receiver

Worst-Case Source Failure Mode	**Desired Worst-Case Failure Effect**	**Number of Sources Required (n)**	**Error Containment Interface Function**
Failed Passive	Operational	F + 1	Select
Failed Active	Failed Passive	2F	Vote
Failed Active	Operational	2F + 1	Vote

2.6.7. Error Mitigation for Multiple Receivers

Figure 12 illustrates a configuration for redundant sources sending to multiple receivers. The failure modes for the sources now add the dimension of symmetry as perceived by the receivers. The resulting failure modes are listed in Table 2. Note that these are worst-case failure constraints and that the implication concept applies (e.g., active failure constraint implies that the failure can actually be passive, and asymmetric failure can actually be symmetric). The desired worst-case effects now include consideration of agreement (i.e., symmetry) among the receivers. Error mitigation in this configuration is a combination of line failure-mode mitigation (i.e., relative to the passive-versus-active failure mode dimension) and asymmetry mitigation (i.e., relative to the symmetric-versus-asymmetric failure mode dimension). The error mitigation parameters for various combinations of worst-case source failure mode and desired worst-case effects at the receivers are listed in Table 3. These configuration parameters are determined assuming that at most F sources are failed and that the outputs of operational sources agree (i.e., they have either exact or approximate equality). Note that, in general, disagreement among the receivers is not desirable. An interesting property of these configurations with multiple receivers is that, because of the desired symmetric worst-case effects, the line failure-mode mitigation must be targeted to achieve the least severe possible state. The two cases where this property is relevant are indicated with * and ** in Table 3. For Passive Asymmetric failure mode and Passive Symmetric worst-case effect (indicated by *), it is possible for one receiver to have only Operational inputs, and therefore, the line failure-mode mitigation must be targeted as a worst-case Operational result at all the receivers with a Select ECXF. The same situation exists for Active Asymmetric failure mode and Passive Symmetric worst-case effect (indicated by **), where the ECXF is a vote. Table 4 shows the configuration parameters when the failure modes of the sources are a combination of passive and active failures. In this table, F_{PS}, F_{PA}, F_{AS}, and F_{AA} denote the maximum number of passive symmetric, passive asymmetric, active symmetric, and active asymmetric failed sources, respectively. A failure-mode model with two or more failure-mode types is called a **hybrid model**. Note that passive failures can be mitigated with a simple selection function and that active failure are mitigated with a vote (or comparison) function. When there is a combination of passive and active failure modes, the mitigation function is a **hybrid vote** in which detected passive-failure inputs are excluded from (i.e., not selected as input to) the vote.

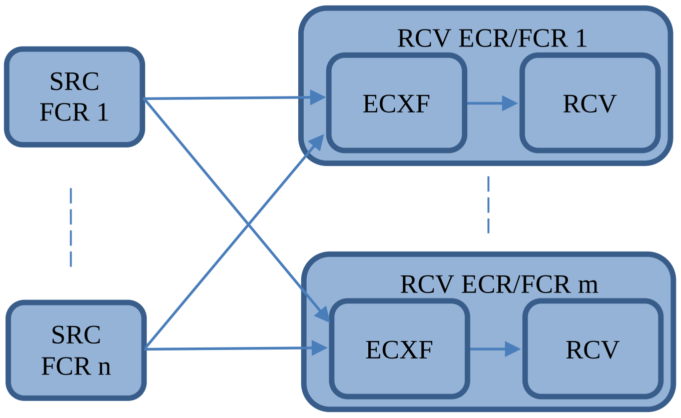


Figure 12: Error Mitigation Configuration for Multiple Receivers

Table 2: Source Functional Failure Modes for Multiple Receiver Configuration

		Symmetry Failure Mode	
		Failed Symmetric (S_m)	**Failed Asymmetric (A_m)**
Worst-Case Line Failure Mode	**Failed Passive (P)**	Failed Passive Symmetric (PS_m)	Failed Passive Asymmetric (PA_m)
	Failed Active (A)	Failed Active Symmetric (AS_m)	Failed Active Asymmetric (AA_m)

Table 3: Error Mitigation Parameters for Multiple Receivers and a Single Worst-Case Failure Mode for the Sources

Worst-Case Source Failure Mode	**Desired Worst-Case Failure Effect**	**Number of Sources, n**	**Error Containment Interface Function**
Passive Symmetric	Operational Symmetric	F + 1	Select
Passive Asymmetric	Passive Symmetric*	F + 1	Select
Passive Asymmetric	Operational Symmetric	F + 1	Select
Active Symmetric	Passive Symmetric	2F	Vote
Active Symmetric	Operational Symmetric	2F + 1	Vote
Active Asymmetric	Passive Symmetric**	2F + 1	Vote
Active Asymmetric	Operational Symmetric	2F + 1	Vote

Table 4: Error Mitigation Parameters for Multiple Receivers and Multiple Failure Modes of the Sources

Passive Failures (F_{PS}, F_{PA})	**Active Failures (F_{AS}, F_{AA})**	**Desired Worst-Case Failure Effect**	**Minimum Number of Sources, n**	**Error Containment Interface Function**
Passive Symmetric, Passive Asymmetric	None	Operational Symmetric	$F_{PS} + F_{PA} + 1$	Select
None	Active Symmetric, Active Asymmetric	Operational Symmetric	$2(F_{AS} + F_{AA}) + 1$	Vote
Passive Symmetric, Passive Asymmetric	Active Symmetric, Active Asymmetric	Operational Symmetric	$2(F_{AS} + F_{AA}) + (F_{PS} + F_{PA}) + 1$	Hybrid Vote

2.6.8. Error Mitigation for Single Source and Multiple Receivers

Figure 13 illustrates a configuration with one source and *m* multiple receivers. We consider two possible failure scenarios. In the first scenario, only the source is failed. In the second scenario, the source and one of the receivers are failed. The receivers must execute a data exchange protocol to mitigate source and receiver failures. The exchange protocol must guarantee that all the non-failed receivers agree on the result and that if the source is not faulty, the result of the exchange is equal to the data from the source. In this configuration, the number of receivers is assumed to be larger than or equal to 2 (i.e., $m \geq 2$)

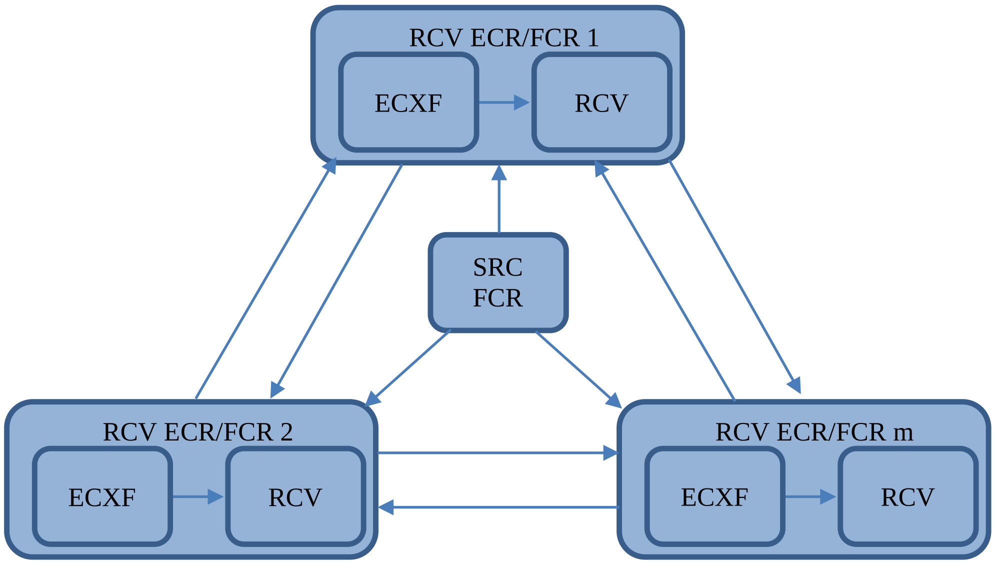


Figure 13: Error Mitigation Configuration for Single Source and Multiple Receivers

Table 5 lists the required configurations for a single-source failure scenario with various worst-case failure modes and desired worst-case results at the receivers. A single receiver-exchange protocol is used here in which the receivers relay to each other the data they received from the source. Notice that the exchange protocol can ensure agreement (i.e., symmetry) in the result, but it cannot mitigate the worst-case line failure mode because there are no alternate sources of data. Also, note that an exchange protocol does not mitigate the worst-case result for passive symmetric and active symmetric source failures. The failure categories in the table are based on worst-case failure constraints, and the actual source failures and results may be less severe than listed in the table. For example, a passive asymmetric source failure would be mitigated to an operational symmetric result if the source were operational rather than failed. Finally, note that the hybrid vote, as introduced in Table 4, in effect encompasses both select and vote functions and its application returns the same results as in Table 5. From this point on in this report, only the hybrid vote function will be used.

Table 5: Single-Source, Multiple-Receivers Configuration Parameters for Source-Only Failure Scenario

Worst-Case Source Failure Mode	**Desired Worst-Case Result**	**Minimum Number of Receivers, m**	**Error Containment Interface Function**
Passive Symmetric	Passive Symmetric	2	Select
Passive Asymmetric	Passive Symmetric	2	Select
Active Symmetric	Active Symmetric	2	Select
Active Asymmetric	Active Symmetric	3	Vote

The second failure scenario involves one failed source and at most one failed receiver. To simplify the description, we only consider the case of *m* = 4 receivers, one of which may be failed. The exchange protocol must guarantee agreement on the result among the non-failed receivers, and the result must equal the data sent by the source if it is not failed. The following protocol is used.

Exchange Protocol:

1. The source node sends its data to all the receivers. If a receiver perceives the source as failed passive, it tags the received data as ERROR1.

   Notes:

   - The source is referred as the *step-1 source*. This source has no additional participation in the protocol.
   - There are four *step-1 receivers*.
   - For steps 2, 3, and 4 the goal is to reach agreement on the data received by each step-1 receiver.

2. Each step-1 receiver (now called a *step-2 source*) relays to the other receivers (now called *step-2 receivers*) the data it received directly from the step-1 source. If a step-2 receiver perceives a step-1 source as failed passive, it tags the received data as ERROR2.

   Notes:

   - Each of the four receivers has data from three step-2 sources.

3. For each step-2 source, each step-2 receiver (now called a *step-3 source*) relays to the other two step-2 receivers (now called *step-3 receivers*) the data received from the step-2 source.

   Notes:

   - For each step-2 source, each step-2 receiver has a set of three data items: the data received directly from the step-2 source and the data relayed by each of the other two step-2 receivers.

4. For each step-2 source, all the step-2 receivers perform a hybrid vote on the data they have received. A vote result of ERROR2 or no-majority means that the step-2 source was failed, and in either case the result is labelled ERROR2.

Note:

- At the end of this step, each step-1 receiver has four data items from the step-1 source: the data received directly from the source, and the three hybrid-vote results on the data relayed by the other step-1 receivers.

5. Each receiver performs a hybrid vote on the data from the step-1 source. ERROR2 data is excluded from the hybrid vote. The vote result is the protocol result. A vote result of ERROR1 or no-majority means that the step-1 source was failed.

The protocol data flow is illustrated in Figure 14, which shows the data flow from the source followed by the data flow from receiver RCV1. Note that the section of the data flow beginning with RCV1 is the same as in the first scenario described above. Table 6 shows the protocol mitigation results for various combinations of source and single receiver failure modes. The protocol guarantees a symmetric result, but it cannot mitigate the line failure mode of the source (i.e., if the source is operational, passive, or active, the worst-case result is operational, passive, or active, respectively).

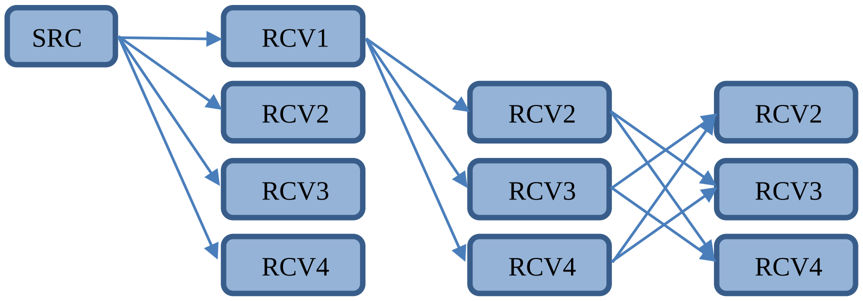


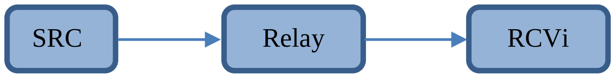
Figure 14: Partial Data Flow of Exchange Protocol for Single Failed Source and Single Failed Receiver

Table 6: Mitigation Results for Single Source, Multiple Receivers Configuration with Combinations of One Source Failure and One Receiver Failure

Worst-Case Source Failure Mode	**Worst-Case Receiver Failure Mode**	**Worst-Case Result**
Operational Symmetric	Passive Asymmetric	Operational Symmetric
	Active Asymmetric	Operational Symmetric
Passive Symmetric	Passive Asymmetric	Passive Symmetric
	Active Asymmetric	Passive Symmetric
Active Symmetric	Passive Asymmetric	Active Symmetric
	Active Asymmetric	Active Symmetric
Passive Asymmetric	Passive Symmetric	Passive Symmetric
	Active Symmetric	Passive Symmetric
	Passive Asymmetric	Passive Symmetric
	Active Asymmetric	Passive Symmetric
Active Asymmetric	Passive Symmetric	Active Symmetric
	Active Symmetric	Active Symmetric
	Passive Asymmetric	Active Symmetric
	Active Asymmetric	Active Symmetric

To gain insight into the properties of the protocol, we consider the properties of the relay function and of the exchange protocol with a hybrid vote. Table 7 shows the effective line failure mode of a relay. Notice that the relay function outputs ERROR when the input data is perceived as failed passive. Also, notice that the line failure mode of the source flows through the relay only when the relay is in the operational function state. Of course, the effective symmetry state of the relay is determined exclusively by the functional failure mode of the relay itself.

Table 7: Effective Failure Mode for Data Relay



Source State	**Relay State**	**RCVi Data**
Operational	Operational	Operational
	Passive	Passive
	Active	Active
Passive	Operational	ERROR
	Passive	Passive
	Active	Active
Active	Operational	Active
	Passive	Passive
	Active	Active

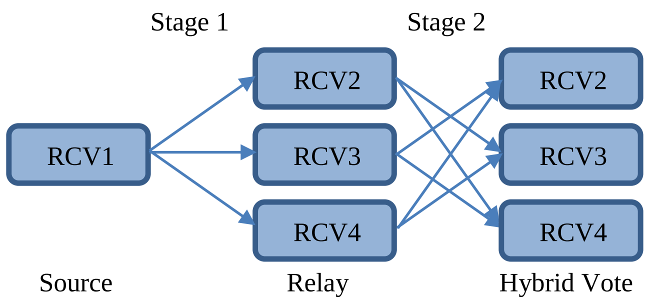


Figure 15: Data Flow for Exchange Protocol of Single-Source, Multiple-Receivers Configuration

Figure 15 illustrates the two-stage exchange-protocol data flow for the single-source, multiple-receivers configuration. We are interested in the symmetry properties of the exchange protocol. We consider cases in which at most one node is failed, which can be the source or a relay. Table 8 breaks down all the cases. In order to achieve agreement, Stage 2 must satisfy at least one of the two following properties:

- The operational relays agree on the data they send out;
- All the relays are symmetric.

The first property guarantees that all the receivers in Stage 2 have the same input majority. The second property guarantees that all the receivers in Stage 2 have the same set of inputs. A symmetric source guarantees the first property. A worst-case failed symmetric relay guarantees the second property. This exchange protocol fails to achieve a symmetric result only when the source and a failed relay are simultaneously asymmetric. To overcome this possibility, the previous configuration shown in Figure 14 has $m = 4$ receivers and the receivers take turns relaying the data received from the source using the exchange protocol described here. This guarantees a symmetric result for each of these source-relaying actions and the same input set to the final vote that decides the result.

Table 8: Symmetry Characteristics of Exchange Protocol for Single-Source, Multiple-Receivers Configuration with Single-Node-Failure Hypothesis

Source Symmetry Mode	**Relay Symmetry Mode**	**Data at Operational Symmetric Relays**	**Applicable Stage Property**	**Data at Operational Voting Receivers**	**Worst-Case Symmetry Result**
Symmetric	All Symmetric	Agreement/ Symmetric	Propagation of Agreement/Symmetry; Propagation by Symmetrics	Majority Agreement; Input Agreement	Symmetric
	At most one Asymmetric	Agreement/ Symmetric	Propagation of Agreement/Symmetry	Majority Agreement	Symmetric
Asymmetric	All Symmetric	Disgreement/ Asymmetric	Propagation by Symmetrics	Input Agreement	Symmetric
	At most one Asymmetric	Disagreement/ Asymmetric	None	Disagreement	Asymmetric

The following section is an overview of the problem of managing groups of redundant nodes.

2.7. Management of Redundant Groups

A function may be realized as a redundant group of nodes in order to achieve a required level of dependability. As shown in the previous section, the mitigation of source errors in line and symmetry failure modes requires source redundancy, failure independence, and agreement among operational sources. Additionally, a function implemented on a data processing system has a set of modes and transitions that must be managed consistently for the members of a redundant group.

Figure 16 illustrates a redundant group of interest (RGOI) that interacts with an input group (IG) and an output group (OG). Mitigation for input and output interactions have been discussed in the preceding sections. The number of nodes in the RGOI depends on the failure modes and dependability requirements of the group.

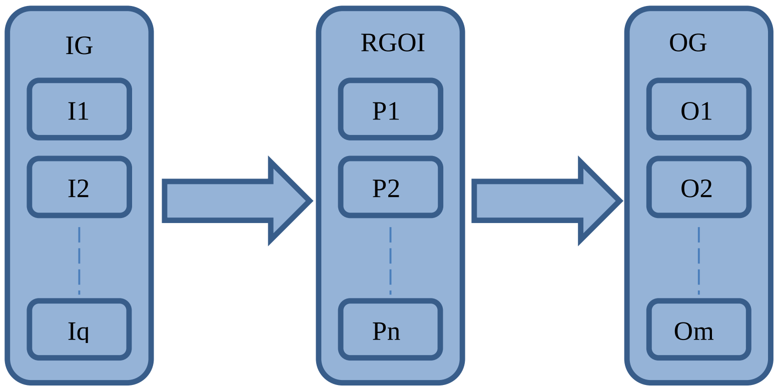


Figure 16: Conceptual View of Redundant Group of Interest Interacting with Input and Output Groups

2.7.1. Local Structure and Modes

A node is essentially a state machine with input, output, and state data and a data processing function. The general structure is illustrated in Figure 17. Figure 18 illustrates a generic mode transition graph for a local processing node. There are four **major modes**: Off, Operation, Failed Passive, and Failed Active. The Operation major mode consists of three **minor modes**: Idle, Recovery, and Operation. When there is a demand for the node, it transitions to the Operation major mode, where it idles until the requirements for successful recovery are satisfied. The node transitions to Operation minor mode when the required conditions are satisfied. The node transitions back to Recovery mode if the requirements for Operation minor mode are violated, and then back to Idle if the Recovery requirements are violated. There are four types of failure: Recoverable Passive, Unrecoverable Passive, Recoverable Active, and Unrecoverable Active. Unrecoverable failures are permanent and require external manual maintenance action to repair the node. Recoverable failure can be automatically repaired, for example, by an internal reset action.

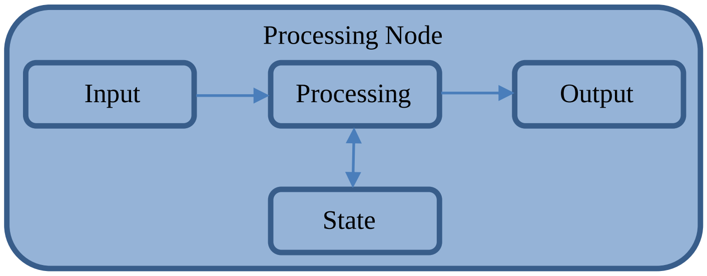
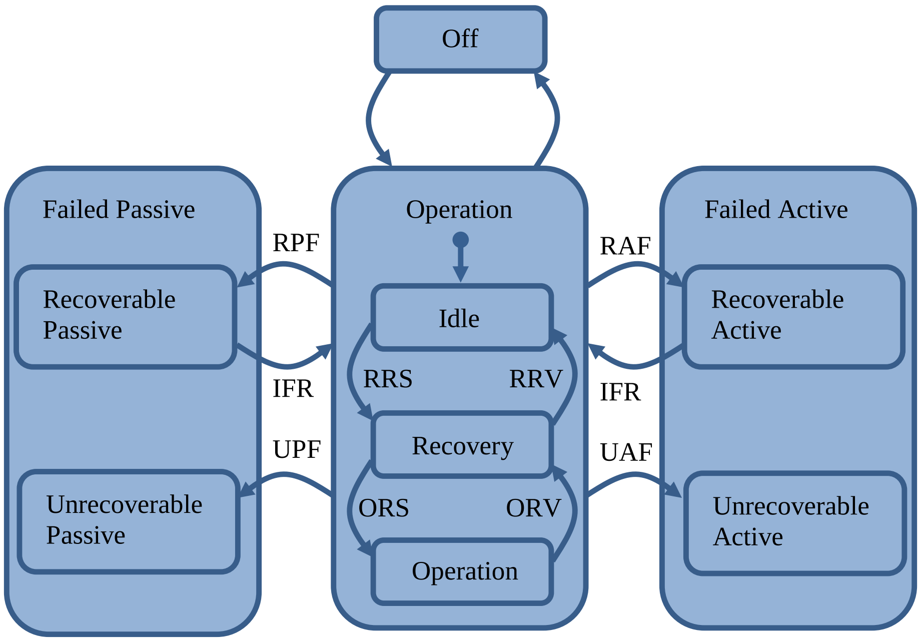


Figure 17: Internal Structure of a Processing Node

Figure 18: Local Mode Graph



RRS	Recovery Requirements Satisfied	RPF	Recoverable Passive Failure
RRV	Recovery Requirements Violated	UPF	Unrecoverable Passive Failure
ORS	Operation Requirements Satisfied	RAF	Recoverable Active Failure
ORV	Operation Requirements Violated	UAF	Unrecoverable Active Failure
		IFR	Internal Failure Recovery

2.7.2. Group Structure and Modes

A redundant group consists of a set of nodes and a communication network. The general structure is illustrated in Figure 19. The nodes operate as a consistent group by executing failure mitigating agreement protocols on one or more of the input, state, and output data sets. The decision of which data sets to execute an agreement protocol upon considers the dependability of the input data, the group dependability requirement, and the architecture-level threat (AT) hypothesis for the group. In concept, the mode graph for the group is the same as for the nodes (see Figure 18). The difference is only in the definition of the modes and transitions as, for example, a group failure occurs when there are certain combinations of failed nodes in the system. In general, a redundant group is failed when good nodes are in disagreement on some or all of their internal data sets (i.e., input, state, and output).

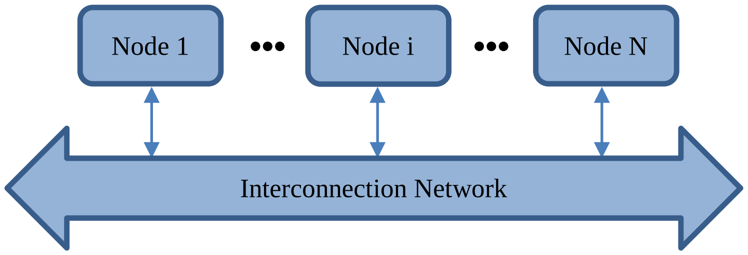


Figure 19: General Structure of a Redundant Group

2.7.3. Dynamic Group Membership

The group management strategy may introduce a dynamic membership capability if the number of nodes in a group is large and the mission duration is long. In this case, the probability of experiencing many node failures, including high severity failures (e.g., failed active asymmetric), is a significant factor in the dependability of the group. Insightful presentations of dynamic group membership can be found in the work by Walter et al. [34], Torres-Pomales et al. [35], and Kopetz [9]. In general, the dynamic membership capability uses local and collective error detection and diagnosis to ascertain the trustworthiness of each node and of the group as a whole. A trusted node is allowed to actively participate with other nodes in the group function. Every node keeps a dynamic list of the nodes it trusts and whose data it uses in local processing. The entries on this membership list are based on local and collective decision about the trustworthiness of nodes. Figure 20 shows a simple two-state concept for the membership status graph. A node is Isolated if it is not trusted, and it is Integrated if it is trusted. The diagnosis policy is normally biased to ensure that a trustworthy node is not distrusted, because that could lead to the logical exhaustion of nodes even if physically good nodes are available. A **clique** is set of mutually trusting nodes. Ideally, a group has at most one operating clique that includes all the good nodes.

Dynamic membership has several weaknesses that could threaten the dependability of a group. All of these weaknesses listed here are due to imperfect coverage in error detection and diagnosis. First, it is impossible to guarantee accurate diagnosis of asymmetric node failures [36]. This means that a clique may fail due to accumulation of failed nodes. Second, if a group develops multiple cliques, the group may not be able to merge back the cliques and it would thus fail as a group because of inconsistency in the outputs. A multiple-clique condition may be self-sustaining because of mutual distrust among the cliques. The fundamental problem with multiple-clique conditions is that unexpected behavior of a source node observed at a receiver can have multiple causes, including source node faults and unexpected internal state at the source node. Because it may not be possible for receivers to distinguish between these causes by monitoring received data flows, observing receivers may not be able to distinguish between a faulty node and a good node with incorrect state. Either way, the source node is not trustworthy to join normal operation with the observing receiver. Third, an isolated node attempting to recover and join a clique may not be able to develop a membership list consistent with that of nodes in the clique. This could result in an inconsistent internal state in the recovering node that is not detected by the clique, thus leading to the node being judged trustworthy and allowed into the clique. The key issue here is that a node with inconsistent state can be as much a threat to a clique as a failed node, and trusting nodes with inconsistent state can cause the failure of the clique. The potential benefits and weaknesses of dynamic membership must be carefully considered to ensure that the overall risk level is acceptable and the dependability of the group is not compromised.

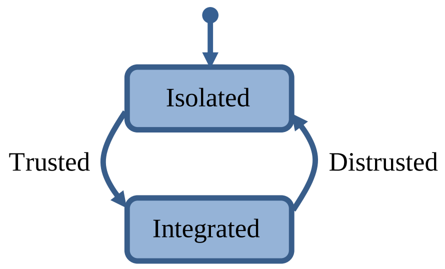


Figure 20: Clique-Relative Membership Status of a Node

2.7.4. Synchrony

A redundant group that performs real-time functions and agreement protocols requires bounded delay uncertainty in data processing and communication, as well as bounded uncertainty in the relative timing of events at different nodes of the group (i.e., bounded relative timing skew). The bound on relative timing uncertainty in effect defines **equality** in the time dimension for redundant data flows in the group. Hence, correctness in a real-time redundant group is determined by **accuracy** (i.e., bound on absolute timing uncertainty) and **precision** (i.e., bound on relative timing uncertainty) requirements that must be guaranteed by the group. Because of the interdependencies among the nodes, these group-level timing requirements translate to node and network timing accuracy and precision requirements.

Actions in a group are triggered by events originating from sources internal or external to the group. The timing of these events may be time-independent or time-referenced. Events can be distributed to all the group nodes with a certain level of accuracy and precision. **Time-independent** sourced events can be the result of internal processing condition (e.g., error exceptions) or they can be derived from external events (e.g., the arrival of input data). **Time-referenced** (i.e., **time-triggered**) **events** can be derived from **logical-time clocks** that keep track of the passage of real time as indicated by clock oscillators [37]. **Synchronized periodic clocks** are re-synchronized periodically using event agreement protocols to ensure a certain logical time precision which is normally much smaller than the period. **Unsynchronized periodic clocks** are allowed to run independently without an agreement protocol, and their precision is approximately equal to their period.

Event timing synchrony and value agreement are critical aspects of a real-time redundant group and serve as a basis for inline and cross-lane error detection in input, state, and output data flows.

2.8. Safety-Risk Mitigation Limitations

The effectiveness and confidence in the proposed general safety-risk mitigation strategy is limited by uncertainties in the accuracy of models of the threats and the system implementation, and by uncertainties in the correctness of analyses of models (i.e., differences between models and reality). These uncertainties may be due to simplifying abstractions in the models to enable analysis tractability, or due to lack of knowledge about the threats, the system, or the environment. The degree of uncertainty is related to the complexity of the threats, the system, and the environment. Though in principle the system properties are based on assume-guarantee relations, in reality for highly complex systems it may not be possible to ensure that these properties and relations are valid for all possible, or even all assumed, conditions experienced by the system. The coverages (i.e., probability of being true) of the fault hypothesis and the error mitigation mechanisms may be less than perfect. This coverage uncertainty is a critical factor in the design of safety-critical systems because dependability is determined by these coverages. Coverage uncertainties can be mitigated by a defense-in-depth strategy that adds robustness to the fault hypothesis and the error mitigation capabilities. The resulting system may be dependable but sub-optimal (i.e., over-designed) relative to the amount of resources and overall complexity. Without more precise information to reduce uncertainties (i.e., increase confidence) about the properties of the system and the operating conditions, the only other option to ensure dependability is to make the system properties more robust to uncertainties (i.e., less sensitive, more tolerant).

2.9. Robust Resilience

Siewiorek et al. [38] define **robustness** as the ability of a system to identify and handle errors (at the functional inputs or internal to the system) of varying severity in a consistent and predictable manner. Kopetz in [9] states that a system is robust if the severity of the consequences of a fault is inversely proportional to the probability of fault occurrence (i.e., frequent faults have less severe effects on service quality than infrequent faults). Bishop et al. [39] considers a system to be robust if it resists a wide range of attacks (i.e., faults) and adverse operational conditions without significant service degradation, but may not have the ability to restore lost functionality (i.e., service quality). The main aspect of interest regarding system robustness is the assessment of service quality over a wide range of operational conditions (including number and severity of internal faults). Under these conditions, a service is robust if it satisfies Kopetz's criterion for robustness (i.e., severity of effects is inversely proportional to the frequency of the fault). In general, robustness does not imply ability to recover the service, but recovery may be essential for safety-critical real-time systems in order to satisfy Kopetz's criterion.

A resilient system may experience degraded operation due to faults, but will eventually recover. In [40], Laprie defines resilience as "the persistence of service delivery that can justifiably be trusted, when facing changes." Trivedi et al. [41] states that "resilience deals with conditions that are outside the design envelope" and, in general, refers to the ability of a system to resist and recover from shock or strain. In [42], Leveson states that resilience is often defined as "the ability to continue operations or recover stable state after a major mishap or event". Bishop et al. [39] states that "a resilient system is effectively a survivable system that is capable of restoring not only its performance level back to desirable levels, *but also the capacity of the system itself to recover*, maintaining its ability to sustain future attacks or failures." From our perspective, resilience describes the ability of a system to mitigate the effects of component-level service degradations. A system is resilient to faults if its service quality is hard to degrade and quality is restored after the fault condition has subsided. For safety-critical real-time systems, availability (in terms of reliability and recoverability) and integrity are the most significant measures of service quality. Therefore, for safety-critical real-time systems, we define **resilience** as the ability to preserve and restore service availability and integrity under stated conditions.

Robust resilience is a defense-in-depth concept to mitigate the risk (i.e., causes and effects) of faults in a system. The fault hypothesis divides the fault space into *normal* and *rare* fault subsets based on assumed FCR failure rates [9]. The normal fault subset may also be referred as *expected* or *credible*, and the rare fault subset may be referred as *unexpected* or *non-credible*. A primary system design goal is to achieve fault-handling coverage as close as possible to 100% for normal faults. To mitigate the risk of a fault assumption violation, the system fault-handling design should also cover a significant percentage of the most likely fault scenarios in the rare-fault region. According to Kopetz [26], in a properly designed system, a likely scenario for a fault-hypothesis violation is a transient correlated failure of multiple FCRs. For such scenarios, a robust safety-critical real-time system should recover to an operational state with high probability.

A system should be designed to meet the functional and service quality requirements while handling the fault space defined by the fault hypothesis. The operational fault-handling effectiveness of a system depends on two basic factors: the **fault-assumption coverage** (i.e., the probability that actually occurring faults are within the assumed fault space) and the **fault-handling coverage** (i.e., the probability that assumed faults are properly handled by the system) [13] [43]. System development is an iterative risk optimization process involving refinements to the fault hypothesis and the fault handling strategy and mechanisms. Usually, the fault modes and fault rates of non-redundant, primitive system components are fixed as determined by the implementation technology. The component fault rate is a determining factor in the amount of redundancy needed to satisfy the system service availability requirement, and the fault modes (i.e., failure semantics) influence the amount and organization of redundancy to satisfy the integrity requirement [44] [45] [27] [43]. In general, to satisfy particular availability and integrity requirements, higher fault rates necessitate increased redundancy, while less constrained fault modes demand increased redundancy and more complex organization. However, using redundancy in hardware, software, time and/or information domains [46], it is possible to define higher-level structural components with less severe (i.e., safer) failure modes and failure rate profiles [47]. This approach increases the complexity of these higher-level components in exchange for easier-to-handle component failure modes and failure rates, which enables a simpler high-level system design. Examples of this include Honeywell's SAFEbus with self-checking-pair components [48], Airbus' Command-Monitor (COM-MON) computers [49] [19] [21], and the Boeing 777 primary flight computers with triple internal redundancy in a command-monitor-standby configuration [50] [51] [52] [53]. The design of the Boeing 777 flight computers shows that it is possible to constrain the failure-mode rates of high-level components while preserving availability.

2.10. Other Sources in the Literature

There any many other sources of knowledge and insight into safety-risk mitigation and fault tolerance. Heimerdinger and Weinstock have proposed a conceptual framework for system fault tolerance covering requirements, fault tolerance concepts, and mechanisms [54]. Butler has written an excellent primer on architecture-level fault tolerance [55]. Hussey and Atchison have published an overview of architecture-level design principles for safety-critical systems [56]. Johnson has presented a review of fault management techniques for safety-critical avionics systems [57]. Hammett published a very insightful paper into fault-tolerance avionics requirements and common architectures based on component failure modes [58]. Hasson and Crotty described Boeing's safety assessments processes for commercial airplanes [59], and Yeh has authored a number of papers on the flight control computers for the Boeing 777 transport aircraft [51] [50] [52] [53]. Bleeg presents a very interesting list and description of avionics architecture considerations for commercial transport aircraft [60]. Rushby has published a survey and taxonomy of critical system properties from different system design perspectives and traditions, including dependability, safety engineering, security, and real-time operation [61]. Hatton considers the problem and trade-off in designing one good software version versus multiple redundant versions (i.e., reduction of error likelihood versus reduction of error consequences) [62]. It is not clear within the safety-critical avionics community, including certification authorities, that relying entirely on software development assurance for a single software version will adequately mitigate software safety risks [63] [2]. NASA has published an extensive software safety guidebook [64]. The problem of how to objectively justify and ensure failure independence among redundant functions, systems, and components remains open, especially for safety-critical systems, despite the extensive research and guidance material on the subject [65] [66] [67] [68] [69] [70]. Suri, Walter, and Hugue have published a compendium of select papers on ultra-dependable systems, including several papers on classical designs for ultra-dependable systems [71]. Hall and Driscoll have published an insightful overview of considerations for safety-critical distributed systems [72]. This is but a sample of the numerous treatises concerning fault tolerance and safety risk mitigation.

3. Final Remarks

This report is intended to be part of a design guide for safety-critical computer-based systems. It describes a general strategy to mitigate the risks of threats and achieve the desired level of dependability in safety-critical computer-based systems. The limitations of the strategy were discussed and options to overcome these limitations were presented. The next document in this series will collect all the contributions generated to date on various aspects of the system design problem.

References

[1] S. C. Beland, "Assuring a Complex Safety-Critical Systems of Systems," *AeroTech Congress & Exhibition, SAE Technical Paper 2007-01-3872*, September 2007.

[2] D. Jackson, M. Thomas and L. I. Millett, Eds., *Software for Dependable Systems: Sufficient Evidence?,* Committee on Certifiably Dependable Software Systems; National Research Council, 2007.

[3] National Research Council, Steering Committee for the Decadal Survey of Civil Aeronautics, *Decadal Survey of Civil Aeronautics: Foundation for the Future,* 2006.

[4] *National Aeronautics Research and Development Plan,* Office of Science & Technology Policy, 2010.

[5] D. C. Winter, *Statement before a hearing on Networking and Information Technology Research and Development (NITRD) Program,* Committee on Science and Technology, U.S. House of Representatives, 2008.

[6] A. Mozdzanowska and R. J. Hansman, *System Transition: Dynamics of Change in the US Air Transportation System,* International Center for Air Transportation, Massachusetts Institute of Technology, Report Number ICAT-2008-3, 2008.

[7] Joint Planning and Development Office (JPDO), *NextGen Integrated Development Plan, Version 1.0,* 2008.

[8] Aerospace Vehicle Systems Institute (AVSI), *AFE #58 Summary Final Report,* System Architecture Virtual Integration (SAVI) Program, Document SAVI-58-00-01, version 1, 2009.

[9] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, 2nd ed., Springer Science + Business Media, 2011.

[10] N. Suri, C. J. Walter and M. M. Hugue, "Introduction," in *Advances in Ultra-Dependable Distributed Systems*, IEEE Computer Society Press, 1995, pp. 3 - 15.

[11] M. Paulitsch, J. Morris, B. Hall, K. Driscoll, E. Latronico and P. Koopman, "Coverage and the use of cyclic redundancy codes in ultra-dependable systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2005)*, 2005.

[12] J. R. Pimentel, *An Architecture for a Safety-Critical Steer-by-Wire System,* 2004.

[13] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing,* Vols. 1, No. 1, pp. 11 - 33, January - March 2004.

[14] SAE International, *Aerospace Recommended Practice: Guidelines for the Development of Civil Aircraft and Systems (ARP4754), Revision A,* 2010.

[15] RTCA, Inc., *Design Assurance Guidance for Airborne Electronic Hardware (RTCA/DO-254),* 2000.

[16] RTCA, Inc., *Software Considerations in Airborne Systems and Equipment Certification (RTCA/DO-178C),* 2012.

[17] RTCA, Inc., *Environmental Conditions and Test Procedures for Airborne Equipment (RTCA DO-160G),* 2010.

[18] SAE International, *Aerospace Recommended Practice: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment (ARP4761),* 1996.

[19] W. Torres-Pomales, "Software Fault Tolerance: A Tutorial," NASA Langley Research Center, Technical Report NASA/TM-2000-210616, 2000.

[20] C. Favre, "Fly-by-wire for commercial aircraft: the Airbus experience," *International Journal of Control,* vol. 59, no. 1, pp. 139 - 157, 1994.

[21] P. Traverse, I. Lacaze and J. Souyris, "Airbus Fly-By-Wire: A Total Approach to Dependability," in *Building the Information Society: IFIP 18th World Computer Congress*, Toulouse, France, 2004.

[22] D. Briere and P. Traverse, "AIRBUS A320/A330/A340 Electrical Flight Controls - A family of fault-tolerant systems," in *The Twenty-Third International Symposium on Fault-Tolerant Compuiting (FTCS-23), Digest of Paper*, Toulouse, France, 1993.

[23] G. F. Bartley, "Boeing B-777: Fly-By-Wire Flight Controls," in *The Avionics Handbook*, C. R. Spitzer, Ed., 2001.

[24] M. Abramovici, M. A. Breuer and A. D. Friedman, Digital Systems Testing and Testable Design, IEEE, 1990.

[25] A. M. Saleh, J. J. Serrano and J. H. Patel, "Reliability of Scrubbing Recovery-Techniques for Memory Systems," *IEEE Transactions on Reliability,* vol. 39, no. 1, pp. 114 - 122, April 1990.

[26] H. Kopetz, "On the Fault Hypothesis for a Safety-Critical Real-Time System," in *Lecture Notes in Computer Science - Automotive Software – Connected Services in Mobile Networks*, 2006.

[27] J. H. Lala and R. E. Harper, "Architectural Principles for Safety-Critical Real-Time Applications," *Proceedings of the IEEE,* vol. 82, no. 1, pp. 25 - 40, January 1994.

[28] J. H. Lala, R. E. Harper and L. S. Alger, "A Design Approach for Ultra-Reliable Real-Time Systems," *Computer - Special Issue on Real-Time Systems,* vol. 24, no. 5, pp. 12 - 22, May 1991.

[29] R. Obermaisser, "Fault and Error Containment of Gateways in Distributed Real-Time Systems," *Journal of Software,* vol. 4, no. 7, pp. 686 - 695, 2009.

[30] R. Obermaisser and P. Peti, "A Fault Hypothesis for Integrated Architectures," in *20006 International Workshop on Intelligent Solutions in Embedded Systems*, 2006.

[31] R. Obermaisser and P. Peti, *The Fault Assumptions in Distributed Integrated Architectures,* 2007.

[32] R. J. Abbott, "Resourceful Systems for Fault Tolerance, Reliability, and Safety," *ACM Computing Survey,* vol. 22, no. 1, pp. 35 - 68, March 1990.

[33] J. F. Meyer and R. J. Sundstrom, "On-Line Diagnosis on Unrestricted Faults," *IEEE Transactions on Computers,* Vols. C-24, no. 5, pp. 468 - 475, May 1975.

[34] C. J. Walter, P. Lincoln and N. Suri, "Formally Verified On-Line Diagnosis," *IEEE Transactions o Software Engineering,* vol. 23, no. 11, pp. 684 - 721, November 1997.

[35] W. Torres-Pomales, M. R. Malekpour and P. S. Miner, "ROBUS-2: A Fault-Tolerant Broadcast Communication System," NASA Langley Research Center, Techical Report NASA/TM-2005-213540 , 2005.

[36] K. Shin and P. Ramanathan, "Diagnosis of processors with Byzantine faults in a distributed computing system," in *17th Fault Tolerant Computing Symposium (FTCS 17)*, 197.

[37] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM,* vol. 21, no. 7, pp. 558 - 565, 1978.

[38] D. P. Siewiorek, J. J. Hudak, S. Byung-Hoon and Z. Segal, "Development of a Benchmark to Measure System Robustness," in *The Twenty-Third International Symposium on Fault-Tolerant Computing (FTCS-23), Digest of Papers* , 1993.

[39] M. Bishop, M. Carvalho, R. Ford and L. M. Mayron, "Resilience is More than Availability," in *Proceedings of the 2011 Workshop on Network Security Paradigms (NSPW '11)*, 2011.

[40] J. C. Laprie, "From Dependability to Resilience," in *30th IEEE/IFIP International Conference on Dependable Systems and Networks*, Anchorage, Alaska, 2008.

[41] K. S. Triverdi, D. S. Kim and R. Ghosh, "Resilience in Computer Systems and Networks," in *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD '09)*, 2009.

[42] N. Leveson, N. Dulac, D. Zipkin, J. Cutcher-gershenfeld, J. Carroll and B. Barrett, "Engineering Resilience into Safety-Critical Systems," in *Resilience Engineering: Concepts and Precepts*, E. Hollnagel, D. D. Woods and N. Leveson, Eds., Ashgate Publishing, 2006, pp. 95 - 124.

[43] D. Powell, "Failure Mode Assumptions and Assumption Coverage," in *Twenty-Second International Symposium on Fault-Tolerant Computing (FTCS-22),* Boston, MA, 1992.

[44] M. H. Azadmanesh and R. M. Kieckhafer, "New Hybrid Fault Models for Asynchronous Approximate Agreement," *IEEE Transaction on Computers,* vol. 45, no. 4, pp. 439 - 449, April 1996.

[45] M. H. Azadmanesh and R. M. Kieckhafer, "Exploting Omissive Faults in Synchronous Approximate Agreement," *IEEE Transactions on Computers,* vol. 49, no. 10, pp. 1031 - 1042, October 2000.

[46] B. W. Johnson, "An Introduction to the Design and Analysis of Fault-Tolerant Systems," in *Fault-Tolerant Computer System Design*, Prentice Hall, 1996, pp. 1 - 87.

[47] F. Cristian, "Understanding Fault-Tolerant Distributed Systems," *Communications of the ACM,* vol. 34, no. 2, pp. 57 - 78, February 1991.

[48] K. Driscoll, B. Hall, H. Sivencrona and P. Zumsteg, "Byzantine Fault Tolerance, from Theory to Reality," in *The 22nd International Conference on Computer Safety, Reliability and Security SAFECOMP*, 2003.

[49] D. Briere, C. Favre and P. Traverse, "Electrical Flight Controls, From Airbus A320/330/340 to Future Military Transport Aircraft: A Family of Fault-Tolerant Systems," in *The Avionics Handbook,* C. R. Spitzer, Ed., CRC Press, 2001, p. Chapter 12.

[50] Y. C. Yeh, "Design Considerations in Boeing 777 Fly-By-Wire Computers," in *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium*, 1998.

[51] Y. C. Yeh, "Triple-Triple Redundant 777 Primary Flight Computer," in *Proceedings of the 1996 Aerospace Applications Conference*, 1996.

[52] Y. C. Yeh, "Safety Critical Avionics for the 777 Primary Flight Controls System," in *20th Digital Avionics Systems Conference (DASC)*, 2001.

[53] Y. C. Yeh, "Unique Dependability Issues for Commercial Airplane Fly by Wire Systems," in *Building the Information Society, IFIP 18th World Congress, Topical Sessions*, 2004.

[54] W. L. Heimerdinger and C. B. Weinstock, "A Conceptual Framework for System Fault Tolerance," Carnegie Mellon University - Software Engineering Institute (CMU/SEI), 1992.

[55] R. W. Butler, "A Primer on Architecture Level Fault Toelrance (NASA/TM-2008-215108)," NASA Langley Research Center, 2008.

[56] A. Hussey and B. Atchison, "Safe Architectural Design Principles," 2000.

[57] D. M. Johnson, "A review of fault management techniques used in safety-critical avionic systems," *Progress in Aerospace Sciences,* vol. 32, no. 5, pp. 415 - 431, October 1996.

[58] R. Hammett, "Design by Extrapolation: An Evaluation of Fault Tolerant Avionics," *IEEE Aerospace and Electronic Systems Magazine,* vol. 17, no. 4, pp. 17 - 25, 2002.

[59] J. Hasson and D. Crotty, "Boeing's Safety Assessment Processes for Commercial Airplane Designs," in *16th Digital Avionics Systems Conference (DASC)*, 1997.

[60] R. J. Bleeg, "Commercial Jet Transport Fly-By-Wire Architecture Considerations," in *9th AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 1988.

[61] J. Rushby, "Critical System Properties: Survey and Taxonomy," SRI International, Technical Report SRI-CSL-93-1, 1993.

[62] L. Hatton, "N-Version Design Versus One Good Version," *IEEE Software,* vol. 14, no. 6, pp. 71 - 76, Nov/Dec 1997.

[63] Federal Aviation Administration, Certification Authorities Software Team (CAST), *Reliance on Development Assurance Alone when Performing a Complex and Full-Time Critical Function. Position Paper CAST-24,* 2006.

[64] National Aeronautics and Space Administration (NASA), "NASA Software Safety Guidebook (NASA-GB-8719.13)," 2004.

[65] J. Downer, "When Failure is an Option: Redundancy, reliability, and regulation in complex technical systems," Center for Analysis of Risk and Regulation, 2009.

[66] V. M. Hoepfer, J. H. Saleh and K. B. Marais, "On the value of redundancy subject to common-cause failures: Toward the resolution of an on-going debate," *Reliability Engineering and System Safety,* vol. 94, no. 12, pp. 1904 - 1916, 2009.

[67] P. T. Popov, L. Stringini and B. Littlewood, "Choosing Between Fault-Tolerance and Increased V&V for Improving Reliability," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, 2000.

[68] R. T. Wood, "Diversity Strategies to Mitigate Postulated Common Cause Failure Vulnerabilities," in *Seventh American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies (NPIC&HMIT 2010)*, Las Vegas, Nevada, 2010.

[69] IAEA, "Protecting against Common Cause Failues in Digital I&C Systems of Nuclear Power Plants," International Atomic Energy Agency (IAEA), 2009.

[70] J. H. Lala and R. E. Harper, "Reducing the Probability of Common-Mode Failure in the Fault Tolerant Parallel Processor," in *12th Digital Avionics Systems Conference (DASC)*, Fort Worth, Texas, 1993.

[71] N. Suri, C. J. Walter and M. M. Hugue, Advances in Ultra-Dependable Distributed Systems, IEEE Computer Society Press, 1995.

[72] B. Hall and K. Driscoll, "Distributed System Design Checklist," NASA Langley Research Center, NASA/CR-2014-218504, 2014.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 11 - 2015	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Overview of Risk Mitigation for Safety-Critical Computer-Based Systems	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Torres-Pomales, Wilfredo	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 999182.02.50.07.02
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20622
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA-TM-2015-218988

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 62
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report presents a high-level overview of a general strategy to mitigate the risks from threats to safety-critical computer-based systems. In this context, a safety threat is a process or phenomenon that can cause operational safety hazards in the form of computational system failures. This report is intended to provide insight into the safety-risk mitigation problem and the characteristics of potential solutions. The limitations of the general risk mitigation strategy are discussed and some options to overcome these limitations are provided. This work is part of an ongoing effort to enable well-founded assurance of safety-related properties of complex safety-critical computer-based aircraft systems by developing an effective capability to model and reason about the safety implications of system requirements and design.

15. SUBJECT TERMS
Computer; Error; Failure; Fault; Model; Safety; System

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	44	(757) 864-9658