AN ABSTRACT OF THE DISSERTATION OF

Hoda Mehrpouyan for the degree of Doctor of Philosophy in Mechanical Engineering presented on July 31, 2014.

Title: A Framework for Assessing and Improving the Resilience of Complex Engineered Systems During the Early Design Process

Abstract approved: ____

Irem Y. Tumer, Christopher J. Hoyle

As modern systems continue to increase in size and complexity, they pose significant safety and risk management challenges. System engineers and much of the government research efforts are focused on understanding the attributes and characteristics that emerge from the interactions of components and subsystems. As a result, the objective of this research is to develop techniques and supporting tools for the verification of the resilience of complex engineered systems during the early design stages. Specifically, this work focuses on automating the verification of safety requirements to ensure designs are safe, automating the analysis of design topology to increase design robustness against internal failures or external attacks, and allocating appropriate level of redundancy into the design to ensure designs are resilient. In distributed complex systems, a single initiating fault can propagate throughout engineering systems uncontrollably, resulting in severely degraded performance or complete failure.

This research is motivated by the fact that there is no formal means to verify the safety and resilience properties, and no provision to incorporate related analysis into the design process.

A Framework for Assessing and Improving the Resilience of Complex Engineered Systems During the Early Design Process

by

Hoda Mehrpouyan

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 31, 2014
Commencement June 2015

ACKNOWLEDGEMENTS

I have had the honor and pleasure of working with Dr. Irem Y. Tumer throughout my Ph.D. years at Oregon State University. Through Dr. Tumer's help and guidance, we have been able to address many challenging topics in the area of safety and resilience of complex systems design and advance the research in this discipline. I would like to thank Dr. Tumer for not just being a supervisor but also for being a great mentor and friend through the challenging but rewarding Ph.D. program. I look forward to continuing my collaboration with her after the completion of my Ph.D. degree. I would also like to thank my co-advisor, Dr. Christopher Hoyle, for invaluable brainstorming and discussions.

I would also like to acknowledge Dr. Guillaume Brat, Dr. Dimitra Giannakopoulou, Dr. Misty Davies, and Dr. Falk Howar for all of your support during my time at NASA. I thank Professor Andy Dong at University of Sydney, who were a great collaborator, and provided me exposure to a wide breadth of research. I would also like to thank Dr. Tolga Kurtoglu at Palo Alto Research Center for his contribution and help during the process. Special thanks go to all of the members (past and present) of the Complex Engineered Systems Design Lab for their valuable research feedback and support, including Brandon Haley, Joe Piacenza, Douglas Van Bossuyt, David Jensen, Matthew McIntire, Ryan Arlitt, and Scott Proper.

I would also like thank my parents Fatemeh Razzaz Rahmati and Ali Mehrpouyan for being my rock when I needed it the most and for supporting me through every decision I made. Thank you for being the wind beneath my wings, forever giving me inspiration and the encouragement to go on and never give up. I am immensely grateful for your eternal love. My brother, Dr. Hani Mehrouyan for being one of my biggest supporters, for guidance and care and for believing in me throughout my life. My sister-in-law, Sonia Perizzolo for being a wonderful friend and always lending a caring ear. My niece, Mia Mehr, I love you with all my heart and always thankful to have you in my life. Special thanks go to my grandmother Zahra Ghotbzadeh Asrar for being such a strong woman who has provided the necessary encouragement and nurturing throughout my life. This dissertation is written in memory of my grandfathers, Morad Nopour and Hossein Razzaz Rahmati. Your love and your passion for life will remain with me.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

To my parents Fatemeh Razzaz Rahmati and Ali Mehrpouyan, the two most wonderful people in my life. Your endless love and dedication is immeasurable. Your courage in life continues to make me want to be a better person. Thank you and I love you.

Chapter 1: Introduction

The objective of this research is to offer a novel perspective on factors influencing resilience and safety of complex systems. Further, a framework is developed to analyze and evaluate system resilience during early stages of the design process. Specifically, the three research questions of this dissertation are: (1) Can resilience be assessed in the concept of evaluation phase of engineering design through exhaustive requirement verification and other analysis? (2) Can resilience be improved by changing the design topology? and (3) Can a high overhead cost associated with resilience be minimized through enhancing system redundancies? These questions are addressed in the following chapters.

This dissertation is structured using the Manuscript Option: Chapters 3-6 are publications written throughout the last three years, submitted to various journals in Engineering Design. Each manuscript is preceded by a Heading Page that provides information on manuscript title, co-authors, journal name, and submission date. The State of the Art for each manuscript is summarized in the Chapter 2 Literature Review for completeness and ease of reference.

1.1 Motivation

In recent years, technological advancements and a growing demand for highly reliable complex engineered systems, e.g., aviation, power generation, transportation, and health care have made the safety assessment of these systems ever more important. These systems are considered safety-critical and are required to perform more reliably in dynamically uncertain operational environments. Consequently, the development of systems with broader fault detection and diagnosis capabilities is vital for protecting lives, property, and continuity of service. Looking back at the Columbia space shuttle incident of 2003, it is evident that the illogical pursuit of the design mantra of *better, cheaper, faster* led to the decisions that eroded safety without realizing that the risks had dramatically increased [1]. In the aftermath of such catastrophic failure and similar major accidents, it has become apparent that complex systems design and development require both ultra-high safety and high performance.

1.1.1 The Concept of Complex Systems

Every time two or more elements, i.e., human, softwares, and engineered systems, perform together to achieve a common goal, a system is born. If all the components of the system do not perform as expected, the whole system may fail. Hence, even the smallest components in electronic, mechanical, and software systems, may be essential to the success of the system [2].

On the other hand, system of systems (SOS), or a complex system is a collection of other elements which themselves are distinct complex systems that interact with one another to achieve a common goal. The higher the number of systems, the higher the possibility of negative interaction occurrence, that is *emergence*. The reason for this is that (1) the systems constituting the SOS were designed independently and were not originally designed to work together, and (2) each system in the SOS may be of a different technology.

The SOS performs functions and achieves results that can not be achieved by any specific component. More precisely, the performed functions are characterized by the behaviors that are emergent properties of the entire SOS and not the behavior of any specific system component. Pariès [3] categorizes these emergent properties into three distinct types. The subject of emergence has been researched extensively, and three types of emergence are identified: (1) normal emergence, (2) weak emergence, and (3) strong emergence. A system containing all three could be considered to be resilient.

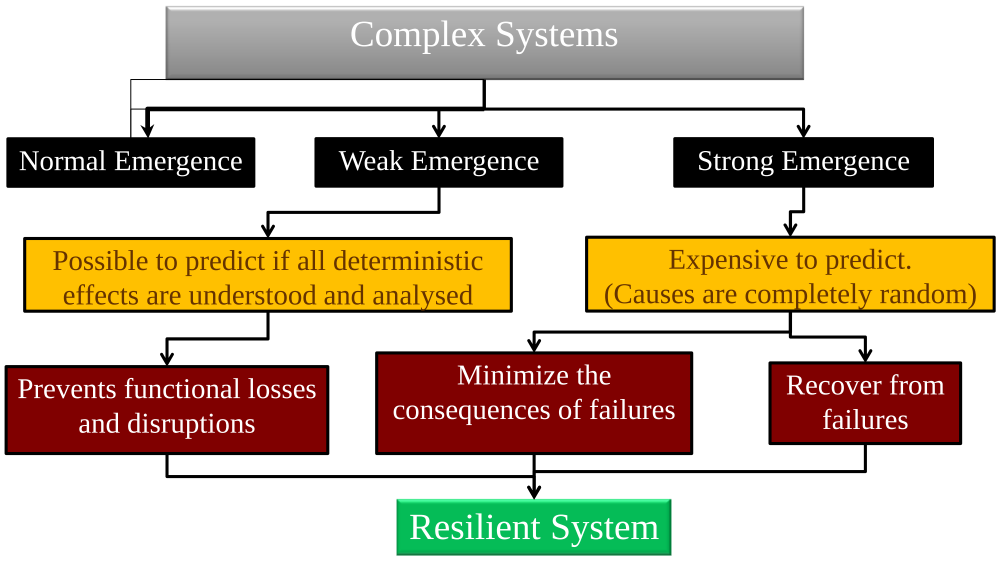

Complex Systems
Normal Emergence
Weak Emergence
Strong Emergence
Possible to predict if all deterministic effects are understood and analysed
Expensive to predict. (Causes are completely random)
Prevents functional losses and disruptions
Minimize the consequences of failures
Recover from failures
Resilient System

Figure 1.1: Resilience and Emergence in Systems.

The three categories of emergence are depicted in Figure 1.1. Normal emergence is a system property, which results from multiple components or sub-systems working together to perform a required function. Hence, normal emergence is a desirable system behavior, while weak and strong emergence are considered undesirable and possibly catastrophic types of emergence. Pavard et al. [4] looks at an air traffic control system as a good example of normal emergence. In this system, Flight Management System (FMS), Instrument Landing System (ILS), and the air traffic controller cooperate together to create an emergent system.

On the other hand, weak emergence is defined as an emergence that could be predicted and prevented if all the laws of physics are taken into the consideration and exhaustive simulation and verification is conducted. One example of weak emergence is the Mars Polar Lander as explained by Leveson [5]. In this accident, the Lander strut vibration was interpreted by software as a landing signal. Therefore, software shut down both engines, and the lander crashed into the planet. If the system, including software, physical system, and their interactions had been verified properly with exhaustive simulation of the vibration of the strut, then the catastrophic failure might never have happened.

Lastly, strong emergence is one of the most difficult types of emergence to be identified. By nature, this type of emergence results from completely random factors and its source is often human. The Nagoya incident as described by Leveson [6] is one example of strong emergence. In this case, the pilot mistakenly sent a wrong command to the flight control system, which resulted in loss of passengers' and pilot's life. The important question to ask here is how the entire system, including aircraft and pilot could have been more resilient and adaptable.

1.1.2 The Concept of Resilience

Madni [7] explains that resilience is a highly broad concept and is used frequently with different meanings in different fields. Depending on the system under design, i.e. business and economy, organizational safety, networks, resilience acquires different meanings. Hutchins [8] defines resilience in the context of economics and business, as the ability of the system to retain prosperity and employment in the face of economic decline caused by losing a powerful type of employer or local business. On the other hand, Luthar et al. [9] describe resilience in terms of network resilience as the ability of the network to continue providing service under adverse conditions. In his research, Luthar et al. [10] define psychological resilience as the ability of person to cope with catastrophe and stress.

Blanchard et al. [11] recognize that resilience engineering is a relatively new concept in the safety engineering discipline, which explicitly identifies and monitors risks. In this concept, risk and failure are viewed as an inability of the system to cope with the internal and external disturbances rather than a component malfunction or sub-system breakdown. In other words, a resilient system is required to recover from internal and external adverse factors including operational environment degradation.

In this research, resilience is viewed as a system's ability to cope with complexity [12] and adapt to changes caused by emergence, either weak or strong. As depicted

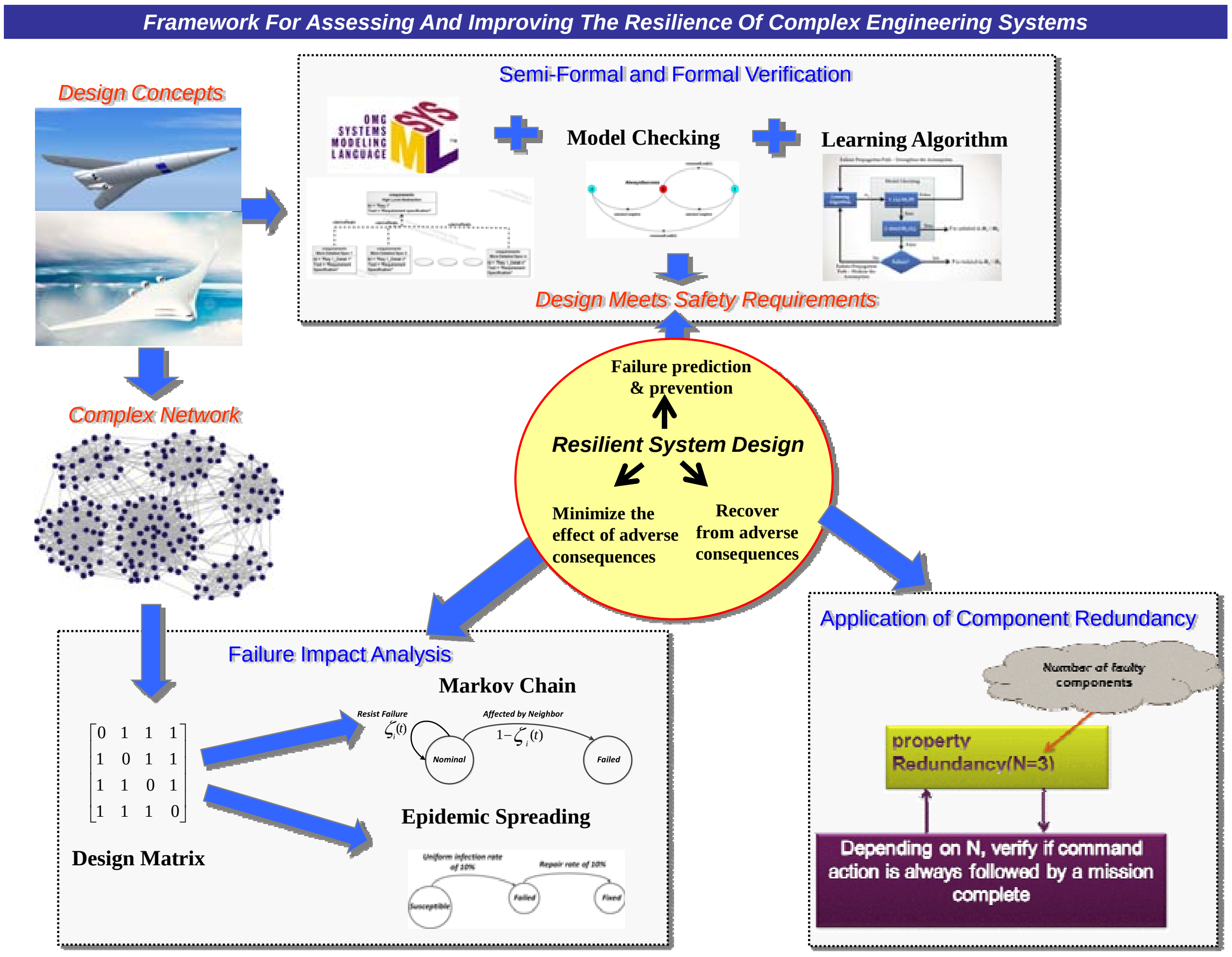

Framework For Assessing And Improving The Resilience Of Complex Engineering Systems
Design Concepts
Semi-Formal and Formal Verification
Model Checking
Learning Algorithm
Design Meets Safety Requirements
Complex Network
Failure prediction & prevention
Resilient System Design
Minimize the effect of adverse consequences
Recover from adverse consequences
Failure Impact Analysis
Markov Chain
Resist Failure
Affected by Neighbor
Nominal
Failed
Epidemic Spreading
Design Matrix
Application of Component Redundancy
Number of faulty components
property Redundancy(N=3)
Depending on N, verify if command action is always followed by a mission complete

Figure 1.2: An Overview of The Proposed Approach.

in Figure 1.1, resilience is characterized as a multi-faceted property of complex systems that (i) predicts and prevents functional losses and disruptions, (ii) minimizes the impact of failures, and (iii) recovers from disturbances. The result of this research provides a basis for incorporating risks into the design process of engineered systems and tools that proactively capture how the internal system's interactions and environmental factors affect system resilience. In particular, this proposal addresses the following three objectives (Fig. 1.2), which will be discussed in detail in Chapters 4, 5, and 6.

1. Failure prevention in system design through effective anticipation of disruption based on exhaustive simulation and verification of safety requirements.
2. Reduction of adverse consequences through identification of a design topology and physical system infrastructure that is more robust against failures.
3. Recovery from disturbance through component redundancies while determining the least number of component redundancies that are required to tolerate and prevent catastrophic system failure.

Contribution of Objective 1: The main technical challenge to prove the correctness of the design with regards to its safety properties lies in the ability to devise reasoning methods and modeling approaches that handle complex behaviors. The assume-guarantee approach which is based on a compositional and hierarchical reasoning combined with a learning algorithm is able to simplify complex design verification problems. The direct outcome of this objective is to automatically generate *assumptions* on the environment where component and (sub)system performances are *guaranteed* under these assumptions. The automatic generation of failure propagation paths enables the system designers to better address the safety issues in the design. In addition, the second aim of this work is to determine whether exhaustive simulation is feasible by utilizing compositional reasoning and automata learning algorithms. These algorithms are used to reduce the number of states to be inspected during verification. This is especially important where exhaustive simulation is too expensive and non-exhaustive simulation can miss critical safety violation.

Contribution of Objective 2: As engineered systems becoming more architecturally complex, the study of their tolerance to failures of components becomes increasingly challenging. Abstracting such systems from the viewpoint of functions and behaviors seems to be an efficient method of modeling and analysis. Therefore, to establish robustness during the conceptual design phase, the second research objective will utilize complex network theory in conjunction with spectral analysis. These methods provide valuable metrics to assess the capacity of systems to perform within the specified performance envelop despite disturbances to their operating environment. Therefore, a general and precise analytical model such as the Non-Linear Dynamical System (NLDS) model that uses a system of probability equations for accurate characterization of failure propagation in complex networks will be used. The outcome of this objective is to identify those design components that are critical to a system and whose failure would cause shutdown of the overall system. In addition, an epidemic spreading model is developed to compare different conceptual design architectures in terms of their resiliency to failure propagation. This can be done by analyzing how a failure propagates through a system and then fixing failed components to inhibit the propagation of the failure. The proposed model will determine the design architecture that is more resilient to specific component failures.

Contribution of Objective 3: The goal of this research is to eliminate a single point of failure in the design, which was identified in Objective 2. This is achieved by adapting mechanisms such as component redundancy and fault tolerance. The application of component redundancy improves system reliability but also adds cost, weight, size, and power consumption. Therefore, it is vital to minimize the number of redundancies. However, traditional methods for allocating redundancies demand complex mathematics and require large number of parameters that are not even available during early stages of the design process. This research utilizes the behavioral specification of each component and subsystem to describe the overall structure of the design. Then, these specifications are analyzed to determine the least number of component redundancies that are required to tolerate and prevent catastrophic system failure.

1.2 Intellectual Merit

This research capitalizes on several gaps in existing literature and studies in the area of resilient system design. In this research, safety is characterized as an emerging behavior of the system that results from interactions among system components and subsystems, including software and humans. This is where designing a resilient system plays a crucial role in developing proactive design practice for exploiting insights on faults in complex systems. In this context, system failure is viewed as an inability of the system to adapt and recover from disruptions, rather than components' and subsystems' breakdown or malfunction.

Importantly, this work presents a framework for assessing and improving the resilience of complex systems during the early design process. The framework comprises failure prediction and prevention techniques, analyzes the effect of design topology on the propagation of failures, and provides methodologies for system recovery from disruptions. The reason for these layers of analysis is to provide the system designers with a set of tools to support them to integrate safety and resilience where it is needed.

1.3 Broader Impact

From the design side, this research offers a uniform platform for both design engineers to verify a system design and for safety analysts to automate specific parts of safety assessment process. The major benefit of the proposed approach is in its ability to tightly integrate safety and design activities. The framework developed in this research can also be considered for safety and reliability analysis of complex systems such as human systems, finances, and power generation platforms.

Potential users of the proposed framework are system designers and architectures designing a complex system. The framework is intended to assist these professionals to explore tradeoffs between different design solutions and evaluate safety and resilience of the system.

1.4 Organization Of Dissertation

Chapter 3 and 4 present a methodology for the verification of safety requirements for design of complex engineered systems. The proposed approach combines a *SysML* modeling approach to document and structure safety requirements, while an *assume-guarantee* technique is used for the formal verification purpose. The assume-guarantee approach, which is based on a compositional and hierarchical reasoning combined with a learning algorithm, is able to simplify complex design verification problems. The objective of the proposed methodology is to integrate safety into early design stages and help the system designers to consider safety implications during conceptual design synthesis, reducing design iterations and cost. The proposed approach is validated on the quad-redundant Electro-Mechanical Actuator (EMA) of a Flight Control Surface (FCS) of an aircraft.

Chapter 5 and 6 describe a graph spectral approach to calculate the resilience of complex engineered systems. The resilience of the design architecture of complex engineered systems is deduced from graph spectra. This is calculated from adjacency matrix representations of the physical connections between components in complex engineered systems. Furthermore, we propose a new method to identify the most vulnerable components in the design and design architectures that are robust to transmission of failures. Non-linear dynamical system (NLDS) and epidemic spreading models are used to compare the failure propagation mean time transformation. Using these metrics, we present a case study based on the Advanced Diagnostics and Prognostics Testbed (ADAPT), which is an Electrical Power System (EPS) developed at NASA Ames as a subsystem for the Ramp System of an Infantry Fighting Vehicle (IFV).

Chapter 7 presents a novel safety specification and verification approach based on compositional reasoning and model checking algorithms. The behavioral specification of each component and subsystem is modeled to describe the overall structure of the design. Then, these specifications are analyzed to determine the least number of component redundancies that are required to tolerate and prevent catastrophic system failure. The framework utilizes Labelled Transition Systems (LTS) formalism to model the behavior of components and subsystems. Furthermore, compositional analysis is used to reason about the components' constraints (or assumptions) on their environments and the properties (or guarantees) of their outputs. A model of quad-redundant Electro-Mechanical Actuator (EMA) is constructed and, in an iterative approach, its safety properties are analyzed. Experimental results confirm the feasibility of the proposed approach for verifying the safety issues associated with complex systems in the early stages of the design process.

Chapter 2: Literature Review

Modern systems such as intelligent vehicle systems, air traffic management systems, and control systems for smart power grids are becoming increasingly more complex and challenging to manage. These systems are considered safety-critical [1] and are required to perform more reliably in dynamically uncertain operational environments. Consequently, the development of systems with broader fault detection and diagnosis capabilities is vital for protecting lives, property, and continuity of service [2, 3, 4].

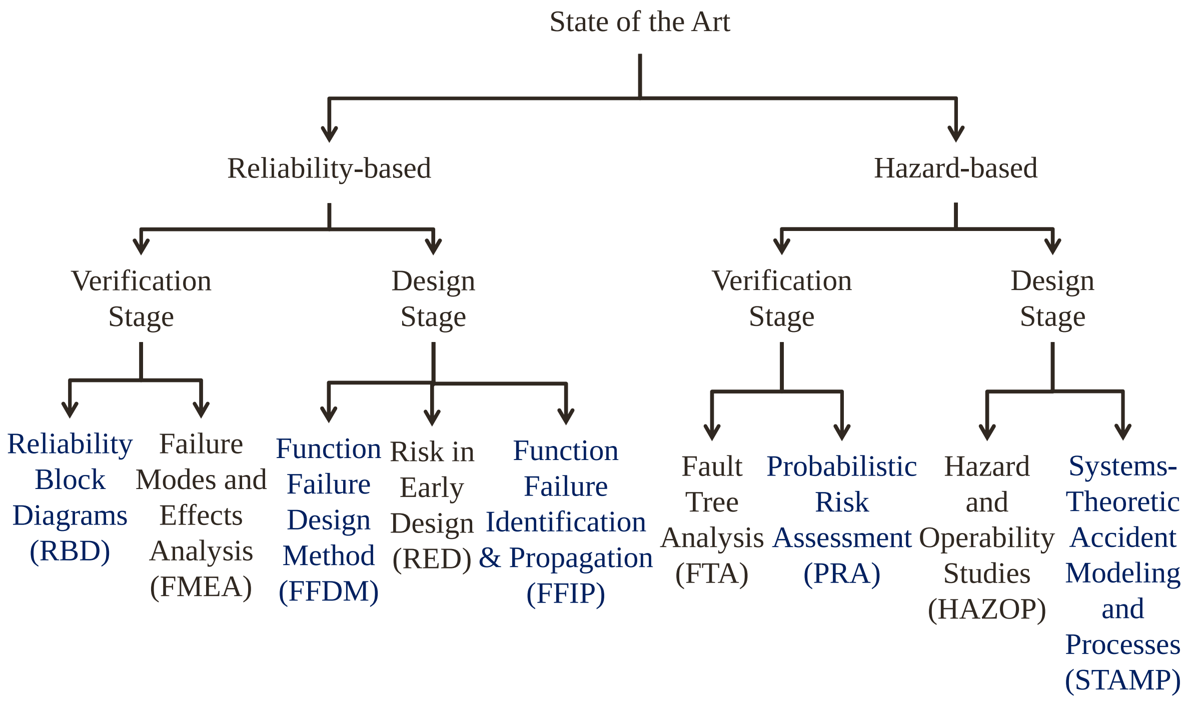

State of the Art
Reliability-based
Hazard-based
Verification Stage
Design Stage
Verification Stage
Design Stage
Reliability Block Diagrams (RBD)
Failure Modes and Effects Analysis (FMEA)
Function Failure Design Method (FFDM)
Risk in Early Design (RED)
Function Failure Identification & Propagation (FFIP)
Fault Tree Analysis (FTA)
Probabilistic Risk Assessment (PRA)
Hazard and Operability Studies (HAZOP)
Systems-Theoretic Accident Modeling and Processes (STAMP)

Figure 2.1: Techniques for Safety and Reliability Analysis of System Design.

A number of failure analysis techniques have been developed over the years. This section reviews several of these common approaches for failure and reliability analysis during conceptual design of engineered systems. As depicted in Figure. 2.1, two distinct categories of methods are usually adopted to address safety analysis of system design. First, reliability-based approaches are based on identifying faults and their likelihood of occurring throughout the system life-cycle. The second category of techniques is based on undesirable system states and focus on identifying paths that reach that state and the likelihood of those paths. Hence, hazard-based techniques are system state centric whereas the reliability-based approaches are fault centric.

This work explores the reliability-based and hazard-based perspectives and incorporates them into the concepts of emergence and resilience as a groundwork for establishing a framework to evaluate the balance between the two areas of risk mitigation. The following section will detail the traditional approach to system design to provide the context of this work.

2.1 Reliability-based Techniques

In order to focus on the design and safety requirements that satisfy the expected level of performance as well as failure and fault tolerance for all expected conditions and environments, reliability and safety analysis techniques are used. Reliability requirements include fault prevention, identification, and isolation. To satisfy these requirements, elimination of hazard, reducing the risk factor associated with failure, and decreasing the impact of failure to an acceptable level are necessary.

2.1.1 Verification Stage Approaches

This group of reliability analysis methods is based on the symbolic logic of the conceptual models of failure scenarios within a design. The goal is to assess the probability of failure occurrence in the system design. One of these methods is the Reliability Block Diagram (RBD) [5], which divides the system into elements based on the functional model of the system design, where each system element is assigned a reliability factor. Then a block diagram of the elements in a parallel, series, or the combination of parallel and series is constructed. Each block represents a function or an event in the system and each element's failure mode is assumed independent from the rest of the system. The reliability factor may or may not be available for all the system design elements and should be assigned by an expert which makes it subjective and hard to validate.

The second popular method of reliability analysis, Failure Mode and Effect Analysis (FMEA) [6] is a bottom up approach that investigates failure modes of components and their effects on the rest of the system. In practice this technique is supported by a top-down analysis to confirm the analytical resolution. FMEA provides an exhaustive analysis to identify the single point of failures and their effects on the rest of the system. The result of the analysis is used to increase reliability, incorporate mitigation into the design, and optimize the design. However, FMEA is very costly in terms of resources, particularly when implemented at the component level within complex systems. Also, occurrences of simultaneous failures and multiple faults are not evaluated. The completeness and correctness of the analysis are very much dependent on the expert knowledge.

2.1.2 Design Stage Approaches

The next category of reliability analysis techniques uses functional modeling to represent the system design for analysis. The Function Failure Design Method (FFDM) introduced by Tumer et al. [7, 8] is an example of such a method. FFDM can be used not only at the early stage of system design but throughout the design process by creating a relationship between system functionality to failure modes and product function to system design concepts. However, FFDM does not determine the "cause" of failure in the design and lacks the analysis to show the severity and detectability rankings of the failures.

Risk in Early Design (RED) method presented by Grantham et al. [9, 10] is built upon the FFDM technique which formulates the functional-failure likeliness and consequence associated with each function failure. Nevertheless, Devendorf [11] attests that RED is not able to assist the designers in effective error proofing during the design process. The knowledge-based repository used by RED to provide relative failure information does not scale to other syntaxes.

In order to overcome these limitations, other research efforts have drawn attention to the importance of failure cascades in reliability analysis. Tumer at al. [12] presented the Function-Failure Identification and Propagation (FFIP) approach for detecting functional failure during the early stages of system design by interleaving failure identification analysis with model based reasoning. The advantage of this approach over other failure analysis methodologies is the fault identification process that is performed at the very early stages of design rather than after-the-fact approaches. In addition, Krus and Grantham-Lough [9] have developed a method based on RED and FFDM for failure propagation mapping that relies on historical failure data and the functional model of the system. This technique analyzes the cascade of a failure along the designed flow path (energy, material, and signal). Further, Hutcheson et al. [13] investigate the changes in component functionality during transition in the midst of critical mission events to analyze the effect of failure. In summary, significant amounts of research and effort has been conducted in order to integrate reliability analysis into the conceptual design stage. However, there is still a need for a theoretically sound and implementable reliability methodology to connect the output of the conceptual design and the reliability data required for reliability modeling.

2.2 Hazard-based Techniques

Hazard-based methodologies focus on system transitions which move from a hazardous state to a failure state based on a set of initiating mechanisms. Therefore, the aim of hazard-based techniques is to identify the potential hazards and the mechanisms and sequences of events which can cause the system to transition to a failure state in the presence of those hazards.

2.2.1 Verification Stage Approaches

Another symbolic logic model is based on the Fault Tree Analysis (FTA) [14] which studies the failure propagation path from the point of start to the vulnerable components and assigns a severity factor to each failure model. One of the benefits of using FTA is its ability to analyze the probability of simultaneous occurrence of failure within a complex systems. On the other hand, the probabilistic evaluation of complex large systems could get computationally intensive.

Also, the correct probabilistic evaluation requires significant amount of resources. Another form of symbolic logic modeling technique is known as Event Tree Analysis (ETA) [15] which differs from FTA analysis in a manner that covers both success and failure events. In this technique all types of events such as nominal system operations, faulty operations, and intended emerging behaviors are modeled. Still, calculating the probability of non-comparative failure or success is difficult to estimate and reach agreement on.

2.2.2 Design Stage Approaches

Another technique for safety analysis is Hazard and Operability Studies (HAZOP) [16] that is based on modeling the interaction flow between components and recognizing a hazard if components deviate from intended operation of design. A set of guidewords are provided to help with identification of such deviations. However, from the context of safety analysis based on interaction between components and their intended environments, HAZOP is unable to produce repeatable hazard analysis of the same accident. The reason for this weakness lies in the highly dynamic and unpredictable nature of interactions between different subsystems and their operational environment. Moreover, depending on the expertise and skills of the safety engineers the deviations can be identified differently.

Systems-Theoretic Accident Modeling and Processes (STAMP). In systematic models such as STAMP, accidents result from several causal factors that occur unexpectedly in a specific time and space [17]. Therefore, the system under consideration is not viewed as a static entity but as a dynamic process that is constantly adapting to achieve its goals and reacting to internal and environmental changes. Consequently, hazards are viewed as complex interactions between system components and their intended environments. The STAMP models are designed based on safety-related constraints and hazards are identified by violation of these safety constraints. There are many benefits in using STAMP models as the basis for hazard analysis of a complex system. However, Johnson et al. [18] state that the STAMP approach has two fundamental weaknesses: the lack of methodological guideline in implementing the constraint flaw taxonomy and the construction of control models in a complex system is complicated. In addition, [18] presents two independent studies of implementing STAMP hazard analysis techniques on the mission interruption of the joint European Space Agency (ESA) and National Aeronautics and Space Administration (NASA) Solar and Heliocentric Observatory (SOHO). The hazard analysis from each study resulted in significantly different conclusion regarding the cause of failure in the system under study.

Chapter 3: Assume-guarantee Verification of Complex Systems based on SysML Functional Requirements

Hoda Mehrpouyan, Irem Y. Tumer, Chris Hoyle, Dimitra Giannakopoulou, Guillaume Brat

International Journal of Prognostics and Health Management

Submitted Jun 2014

3.1 Introduction

In recent years, technological advancements and a growing demand for highly reliable complex engineered systems, e.g., space systems, aircrafts, and nuclear power plants have made the safety assessment of these systems ever more important. Moreover, the growing complexity of such systems has made it more challenging to achieve design solutions that satisfy safety and reliability requirements [1, 2, 3]. Hollnagel et al. [4] recognize the fact that safety violation in complex systems is not necessarily a consequence of components' malfunction or a faulty design. Rather it could be a result of a network of ongoing interactions between all the components and subsystems that introduce undesired behavior. For this reason, Baroth et al. [5] recommends the Prognostic and Health Management System (PHM) as a new technology to replaces the traditional build-in test (BIT) with intelligent prognostics tools to predict the occurrence of unexpected faults. However, given the local safety properties of each component, it is not a trivial matter to infer the safety and reliability of the whole system [6]. Well-specified verification formalism and reasoning tools are needed to study the emerging behavior and to perform exhaustive verification of safety properties. A series of safety standards emerged in recent years that recognize this issue and strongly recommended the use of formal verification methods to control the complexity of safety-critical systems, i.e., the international standard on safety related systems [7] and the SAE & EUROCAE standards in the avionic industry [8, 9]. However, these standards do not specify how to implement formal approaches throughout the design process.

Strategies for engineered system design emerges from a process of requirement decomposition and transforming requirement models into the conceptual models [10, 11]. Requirement models, noted $\boldsymbol{R}$, capture the design problem being solved and conceptual models, noted $\boldsymbol{S}$, represent the specific solution for the design problem. Therefore, the first step in specifying and formulating a complex system is to capture its requirements $\boldsymbol{R}$ and decompose it into the requirements of its sub-systems and components, noted $R = \{R_1, R_2, ..., R_n\}$. The second step is to create a relationship between design requirements and the system that consists of heterogenous sub-systems, i.e., electrical, mechanical, and software ..., noted $S = \{S_1, S_2, ..., S_m\}$. However, this relationship between the set of design requirements and the set of sub-systems and components is a non bijective relationship. A commonly used formalism to address this problem is to focus on *discrete event system dynamics.* This formulation is extended [12, 13, 14] by considering other system features such as structures and functions, so that the predicate ($S_1 \wedge S_2 \wedge ... \wedge S_m \Rightarrow$ Design's Objective) is preserved and satisfied throughout the design process. So the formulation can be summarized as below:

$$\begin{cases} S_i \Rightarrow \{R_k\}_{k\in[1..n]} & S_i \text{ satisfies a sub-set of} \\ & \text{requirements.} \\ \{S_k\}_{k\in[1..m]} \Rightarrow R_i & R_i \text{ satisfied by sub-set of} \\ & \text{ sub-systems or components.} \end{cases}$$

The process of identifying and proving the correctness of these relationships with regards to design safety requirements is the objective of this paper. The remainder of this paper is structured as follows: section *2* discusses the system oriented approaches and their ability in modeling multi-domain complex engineered system and being exploitable for safety analysis. Furthermore, formal verification methods and the definition of *compositional reasoning* and its commonly used terminologies and operators are introduced as a complementary technique to design requirement analysis. In section *3* an overview of the step-by-step implementation of the compositional reasoning algorithm on the components of the design architectures is explained. Further, section *3* outlines the application of the proposed methodology in the analysis and verification of the safety properties of the quad-redundant Electro Mechanical Actuator (EMA) system design. The paper ends with conclusion.

3.2 Related Work

Different standards, e.g., [15, 16] have defined system design as a multidisciplinary collaborative process that defines, develops, and verifies a system solution which satisfies different stakeholders' expectations and meets public safety and acceptability. Therefore, identification and analysis of the system requirements and designing a system according to the identified requirements are the two inter-correlated and complementary processes of system design. While these standards precisely specify the processes involved in the design of a safety critical systems, Lundteigen et al. [17] agree that they do not provide methods and tools for efficient design of complex engineered systems. This highlights the need for appropriate methods and tools to support the integration of safety into the design solution.

3.2.1 SysML for Complex Engineered Systems

Traditional methods and tools used by system engineering are mostly based on a formalism that capture a variety of system features, i.e., requirements engineering, behavioral, functional, and structural modeling, etc. Those with particular focus on requirements engineering are the Unified Modeling Language (UML) [18] to support various aspect of system modeling, Rational Doors [19] to express the requirements, and Reqtify [20] to trace the requirements through design and implementation. UML is developed by the Object Management Group (OMG) in cooperation with the International Council of Systems Engineering (INCOSE). UML is an Object-oriented modeling language that allows hierarchical organization of system component models, which in turn results in easier reuse and maintenance of the system model. However, UML was originally developed for software engineers and its primary application is software-oriented; therefore it does not meet all the system engineer's expectations. For example, UML does not provide a notion to represent continuous flows exchanged within the system, i.e., Energy, Material, and Signal (EMS). The analysis of EMS flows are crucial in system design safety verification for identifying the failure propagation path and identifying the common failure modes. For this reason, the SysML profile was developed borrowing a subset of the UML language to meet the requirements of a general purposed language for system engineering.

SysML is an efficient modeling language for constructing models of complex, multidisciplinary, and large-scale systems. SysML enables the designers of a complex system to model the system requirements, structures, behaviors, and parametric values for a more rigorous description of a system under consideration. Nevertheless, the wide variety of notations provided by SysML lacks formal and detailed semantics required for requirements verification. The goal of this paper is to bridge the gap between semi-formal approaches, e.g., SysML and formal verification methods, e.g., model-checkers to provide the system designers an integrated method to manage and verify the safety properties of complex engineered systems.

3.2.2 Model Checking and Formal Verification

Model checking is one of the approaches to formal verification of finite state hardware and software systems [21, 22]. In this approach, a design will be modeled as a state transition system with a finite number of states and a set of transitions. The design model is in essence a finite-state machine, and the fact that it is finite makes it possible to execute an exhaustive state-space exploration to prove that the design satisfies its requirements. Since there is an exponential relationship between the number of states in the model and number of components that make up the system, the *compositional reasoning* approach is used to handle the large state-space problem. The compositional reasoning technique decomposes the safety properties of the system into local properties of its components. These local properties are subsequently verified for each component. However, Barragan et al. [23] emphasizes the difficulty of transforming the global system requirements into multi-level sub-system and component's local safety properties that need to be verified by a model checker for the design of large scale complex engineered systems. More specifically, the decomposition of complex engineered systems into multi-domain sub-systems in-

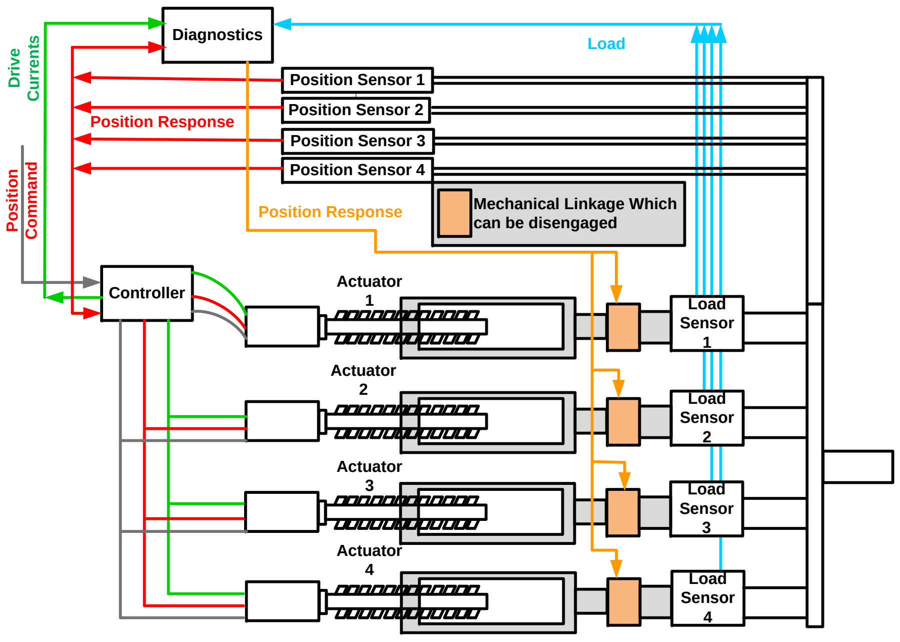


Figure 3.1: Quad-Redundant EMA Scheme.

volving electrical, mechanical, and software components makes the refinement and traceability of the global safety properties very difficult. Therefore, a systematic approach is required to acquire abstract requirements along with safety properties, and map them to system components [24]. Following the work of many researchers, it is concluded that the early stages of system design are the most critical in ensuring that the designed system satisfies its safety requirements [25, 26, 27, 28], this paper aims at addressing this challenge using the system-oriented SysML-based modeling approach combined with formal verification technique.

3.2.3 Case Study

As depicted in Fig. 3.1, a quad-redundant Electro-Mechanical Actuator (EMA) [29] for the Flight Control Surfaces (FCS) of an aircraft, developed in a program sponsored by NASA, is used to illustrate and validate the proposed approach. The positions of the surfaces, A, C, and D, in Fig. 3.2, are usually controlled using a quad-redundant actuation system. The FCS actuation system responds to position commands sent from the flight crew, B in Fig. 3.2, to move the aircraft FCS to the command positions.

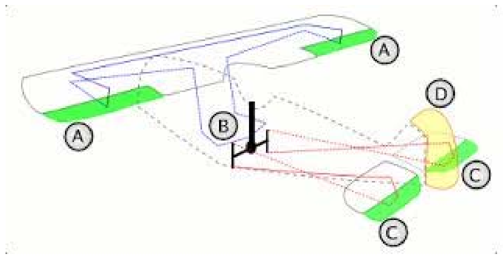


Figure 3.2: Basic Aircraft Control Surfaces.

The EMAs are arranged in a parallel fashion; therefore, each actuator is required to tolerate a fraction of the overall load. To meet safety requirements, each actuator is required to take on the full expected load from the FCS in the extreme case where all three of the four actuators become non-operational. In addition, the design should also consider other issues such as the possibility of the actuators becoming jammed. If one actuator becomes jammed in this parallel arrangement, it will prevent the other ones from moving. Therefore, a mechanism to disengage faulty actuators from the rest of the system is required to avoid the faulty actuators from becoming deadweights. Once an EMA is disengaged from the system it cannot be re-engaged

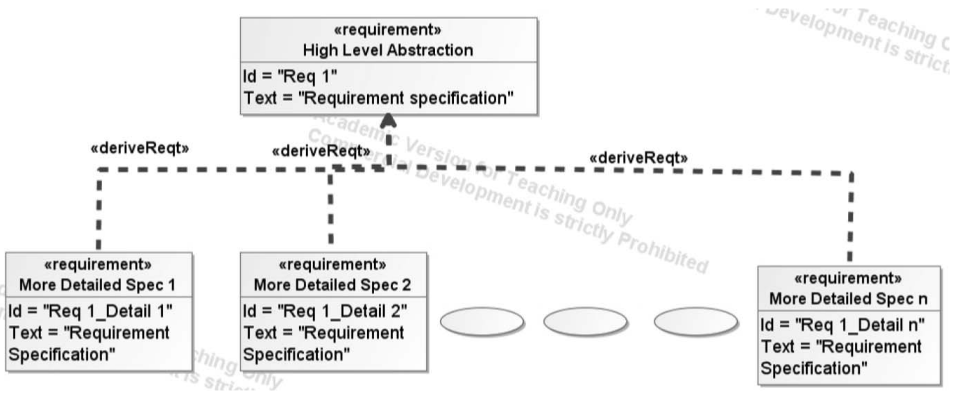


Figure 3.3: Requirements Decomposition.

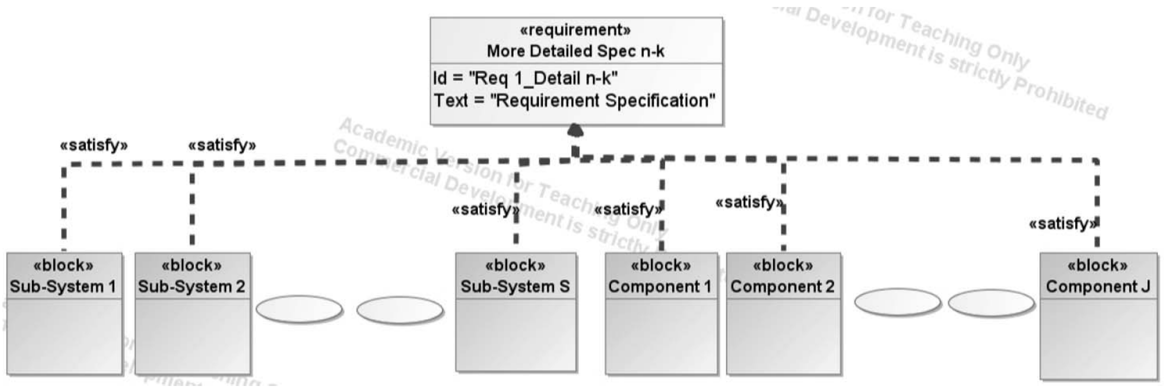


Figure 3.4: Requirements Mapping.

automatically. It is envisioned that this will happen on the ground, once the aircraft has landed.

In order for the design to be reliable, additional redundancies in other components of the system, such as load and position sensors are required. Thus, a fully quad-redundant scheme is envisioned, as depicted in Fig. 3.1. As illustrated, the design features redundancy in the EMAs and the sensor feedback signals. The position command is fed to the control loop, while the load from the FCS is shared by the EMAs. The individual load, current, and position response signals from each EMA are used to perform separate diagnostics on each EMA. Therefore, faults are isolated to the individual actuators, which facilitates adaptive on-the-fly decisions on disconnecting degraded EMAs from the load. A dedicated diagnostics block performs actuator health assessments, and makes decisions on whether or not to disengage any faulty actuators from the flight control surface. The disengagement is made possible by mechanical linkages, which can be disconnected from the output shaft coupling.

3.3 Methodology

Design requirements are the specification of safety constraints initially defined in the design. Requirements are modeled at different levels of abstractions. For example, a higher level of abstraction is used when expressing the global system properties and a low level of abstraction is used when expressing the required features for each system component, i.e. the barriers and materials to be used. Managing this set of specifications is based on iterative decomposition and substitution of the abstract requirements by the requirements that are more concrete.

3.3.1 Safety Requirements Modeling Using SysML

A SysML requirement diagram enables the transformation of text-based requirements into the graphical modeling of the requirements which can be related to other modeling elements. Fig. 3.3 depicts the decomposition of a single abstract requirement into several more explicit ones. A study by Blaise et al. [30] confirms the effectiveness of such diagrams to facilitate the structuring and management of requirements that are traditionally expressed in natural languages.

The next step in the requirement analysis phase consists of mapping the requirements to the corresponding system components or functions. System components are modeled as part of the structural design of a system. The structural design model corresponds to the system hierarchy in terms of systems and subsystems, which are modeled using the Block Definition diagram (BDD). SysML blocks are the best modeling elements to model multi-disciplinary systems and are especially effective during system specification and design. They are effective because blocks are not only able to model logical or physical decomposition of a system, they also enable designers to define specification of software, hardware, or human elements.

Fig. 3.4 illustrates how a single requirement can be satisfied by a set of subsystems and components. The requirement diagram is connected to the structure diagram by a cross connecting element known as *satisfy*. A requirement can be satisfied by a component or subsystem. Furthermore, the detailed modeling of subsystems and components are possible through the use of Internal Block Diagram (IBD). In addition, blocks are a reusable form of description that can be applied throughout the construction of system modeling if necessary. Another advantage of using blocks during the design process is their ability to include both structural and behavioral features, such as properties and operations that represent the state of the system and behavior that the system may display.

Including properties as part of the requirement modeling is specifically important when verifying safety requirements. As Madni. [31] demonstrated, safety is a changing characteristic of complex systems that, once integrated into the design, is not preserved unless enforced throughout system operation. Hollnagel et al. [4] also confirms that safety is a feature that results from what a system does, rather than a characteristic that the system has. Therefore, the proof of safety is provided by the absence of failures and accidents. For this reason, "safety-proofing" a system design is never absolute or complete. Consequently, the proposed approach does not guarantee safe system operation, instead provides formal proof that certain very specific behavioral parameters will be achieved. It is for this reason that in this paper safety is viewed as a system property.

A complete proof of safety is possible through a formal definition of different properties that are linked to each high-level abstract and low-level detailed requirements. Fig. 3.5 represents how a requirement, property, block, and behavioral model are connected to one another. For example, *allocate* as a cross connecting principle in SysML is used to connect a behavior to a component in a structure diagram.

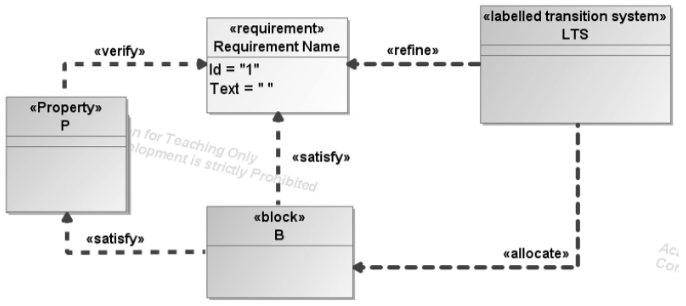


Figure 3.5: Requirements Traceability.

In the proposed approach, individual components' behavior in the system are modeled as Labeled Transition Systems (LTSs), LTSs basically represent a finite state system. The properties of the LTSs make it ideal for expressing the behavioral model of system components. The LTS model is expressed graphically, or by its alphabet, transition relation, and states including single initial state. The LTS of the system is constructed from the LTS of its subsystems, and is verified against safety properties of the design requirements (Fig. 3.5).

3.3.2 Safety Requirements Verification

A model-based verification approach is proposed based on the *behavioral* models of design components, where behavioral specifications are associated with each component. These specifications are then used to analyze the overall design architecture. In this approach, a design will be modeled as a state transition system with a finite number of states and a set of transitions. The design model is in essence a finite-state machine, and the fact that it is finite makes it possible to execute an exhaustive state-space exploration to prove that the design satisfies its requirements. Since there is an exponential relationship between the number of states in the model and number of components that make up the system, the *compositional reasoning* approach is used to handle the large state-space problem. The compositional reasoning technique decomposes the safety properties of the system into local properties of its components. These local properties are subsequently verified for each component. The combination of these simpler and more specific verifications guarantees the satisfaction of the global safety of the overall system architecture design. It is important to note that, the safety requirements of the components are satisfied only when explicit assumptions are made on their environment. Therefore an *assume-guarantee* [32, 33, 34, 35] approach is utilized to model each component with regards to its interaction with its environment, i.e, the rest of the system and outside world.

Since, the LTSs are based on graphical modeling, they can easily become unmanageable for large complex systems. Therefore, an algebraic notation known as Finite State Process (FSP) [36] is used to define the behavior of processes in a design. FSP is a specification language as opposed to a modeling language, with semantics defined in terms of LTSs. Every FSP model has a corresponding LTS description and vice versa. An example FSP and LTS model of the *Electro Mechanical Actuator* (EMA) unit of the quad-redundant EMA of Fig. 3.1 is provided in Table 3.1 and Fig. 3.6 respectively.

Table 3.1: FSP Description of EMA

1: **EMA = (recLoad → ApplyLoad → (allLoadsCompleted → EMA**
2: **| jam → block → Jammed)),**
3: **Jammed = (recLoad → Jammed**
4: **| disengage → unblock → Disengaged),**
5: **Disengaged = (recLoad, allLoadsCompleted, timeout → Disengaged).**

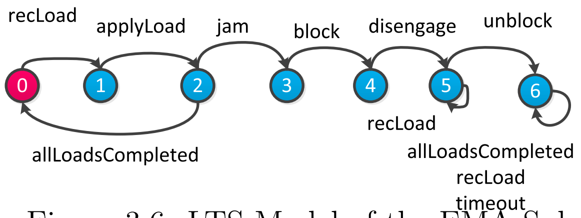


Figure 3.6: LTS Model of the EMA Subsystem.

In the defined model, a EMA receives the load command from the controller and carries out the operation. The Electro Mechanical Actuator is modeled in Table 3.1 with *Jammed* and *Disengaged* as part of its definition. If during the time of maintaining the specified torque or load the EMA functions according to specification, the signal *"all loads are completed"* is sent to the controller. Otherwise, the EMA is considered non-operational or jammed. In the jammed mode, the EMA is incapable of maintaining the required load and prevents the rest of the EMAs from moving. Therefore, it needs to be disengaged from the system.

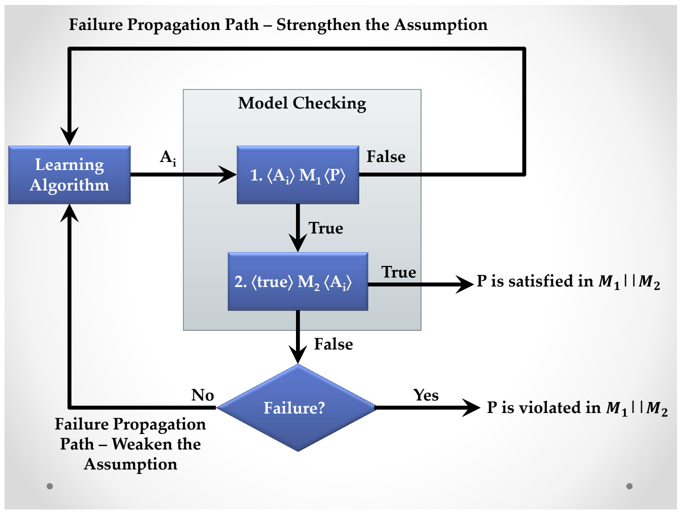
After system modeling, the actual analysis of the models is carried out utilizing the AGR verification technique. In the assume-guarantee methodology, a formula contains a triple $\langle A\rangle\ M\ \langle P\rangle$, where M is defined as a component, P is a safety property, and A is an assumption or constraint on M's environment. The formula is proven correct if whenever M is a component within a system satisfying A, then the system also guarantees P.

The simplest assume guarantee rule for checking a safety property P on a system with two components M_1 and M_2 can be defined as following [37, 35]:
Rule ASYM

$$\begin{array}{rl} 1: & \langle A\rangle\ M_1\ \langle P\rangle \\ 2: & \langle \mathit{true}\rangle\ M_2\ \langle A\rangle \\ \hline & \langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle \end{array}$$

The first rule is checked to ensure that the generated assumption restricts the environment of component M_1 to satisfy P. For example, the assumption A is that there is no Electromagnetic Interference (EMI) or Radio Frequency Interference (RFI) in the environment where component M_1 operates; hence, P is satisfied. The second rule ensures that component M_2 respects the generated assumption. For example, M_2 will not generate any EMI and RFI while operating. If both rules hold then it is concluded that the composition of both components also satisfies property P ($\langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle$).

In this research, the algorithm in [38] is used to automatically generate assume-



Failure Propagation Path – Strengthen the Assumption
Model Checking
Learning Algorithm
A_i
1. ⟨A_i⟩ M_1 ⟨P⟩
False
True
2. ⟨true⟩ M_2 ⟨A_i⟩
True
P is satisfied in $M_1 || M_2$
False
Failure?
No
Yes
P is violated in $M_1 || M_2$
Failure Propagation Path – Weaken the Assumption

Figure 3.7: An Overview of the Algorithm that Generates Assumptions.

guarantee reasoning at the component, subsystem, and system level. The objective is to automatically generate assumptions for components and their compositions, so that the assume-guarantee rule is derived in an incremental manner. The framework of Figure 3.7 depicts the steps involved in performing automated assume-guarantee reasoning while generating the assumptions. If rule (1) is violated, it means that the assumption is too weak, so it does not prevent M_1 from reaching its failure state. Based on the generated failure propagation path, the algorithm *learns* a new assumption with more restriction on the environment which makes the assumption stronger than the previous one. The iteration continues until the first rule of $\langle A \rangle\ M_1\ \langle P \rangle$ is addressed. The next step is to check the second rule $\langle true \rangle\ M_2\ \langle A \rangle$. If the rule still holds, then it is concluded that $\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle$. If the check fails, the algorithm performs analysis on the returned failure propagate path to determine the reason for the failure. If the analysis reveals that A is not the weakest assumption, i.e., elimination of both EMI and RFI is not necessary and only the elimination of EMI suffices to satisfy P, then the learning algorithm will generate a new assumption. If the rules are not satisfied with the generated assumptions, it is concluded that $\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle$ violates the property P.

3.4 Application On The Case Study

In the case study of Fig. 3.2, the Flight Control Surface (FCS) must meet rigorous safety and availability requirements before it can be certified. The FCS has two types of dependability requirements:

- *Integrity*: the FCSs must address safety issues such as loss-of control resulting from aircraft system failures, or environment disturbances.
- *Availability*: the system must have a high level of availability.

Therefore, it is critical for the FCS to continue operation without degradation following a single failure, and to fail safe or fail operative in the event of a related subsequent failure. The movement of the FCS is controlled by a quad-redundant EMAs. A block diagram of the quad-redundant EMAs is depicted in Fig. 3.8. As seen from the figure, the model consists of an EMA block which is an hierarchical representation of four independent EMAs. Each EMA is modeled via the Internal Block Definition diagram (IBD). The individual EMA legs receive the common position command, but act independently of each other and share the flight control surface load among themselves.

Fig. 3.9 depicts a set of high-level requirements. To facilitate the verification process, each level of requirements are associated with a formal Finite State Process

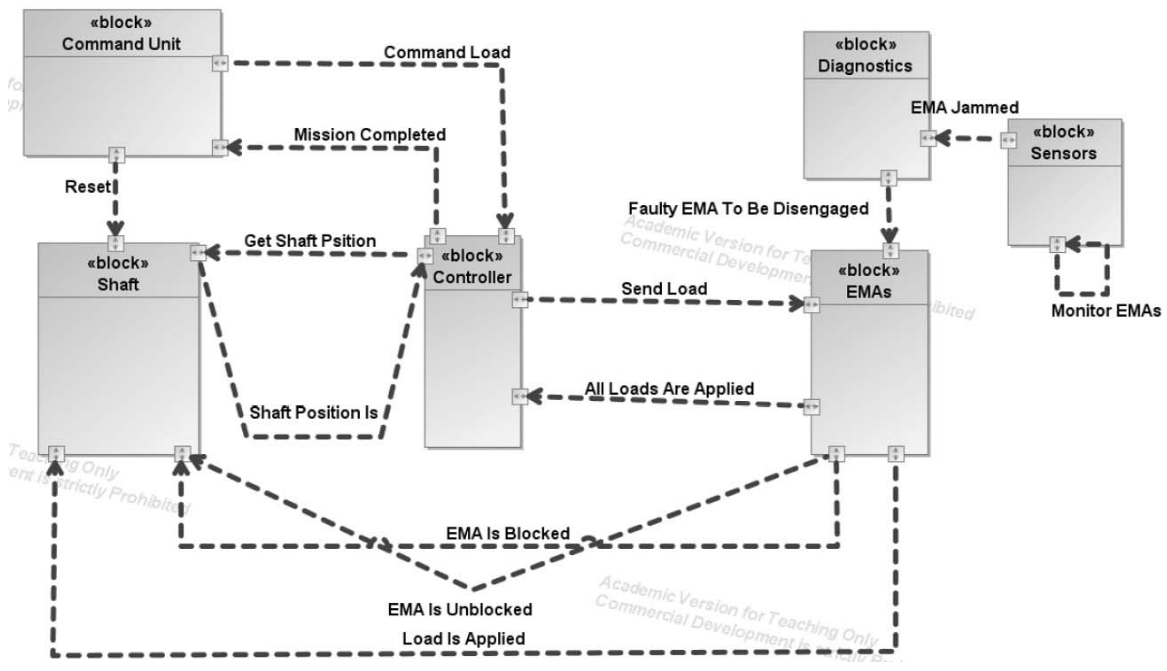


Figure 3.8: Structural Model of the Quad-redundant EMAs.

(FSP) using property stereotype in SysML, meaning that satisfying property *P1* is the same as satisfying properties *P1.1, P1.2, and P1.3*.

The next phase consists of identifying the design architecture (Fig. 3.8), including sub-systems and components to map each requirement to a traceable source. As depicted in Fig. 3.4, requirements mapping are made possible by using the *satisfy* relationship to link a single or set of blocks to one or more requirements. The requirements mapping of quad-redundant EMAs is presented in Table. 3.2.

In order to transform the requirements and the design architecture presented in Fig. 3.8 into a finite model, we use Finite State Process (FSP). As an example, consider the following FSP model of a *controller subsystem* of the quad-redundant EMAs: The controller gets the load command from the command unit and actively regulates the current to each EMA at every time step. The difference between the

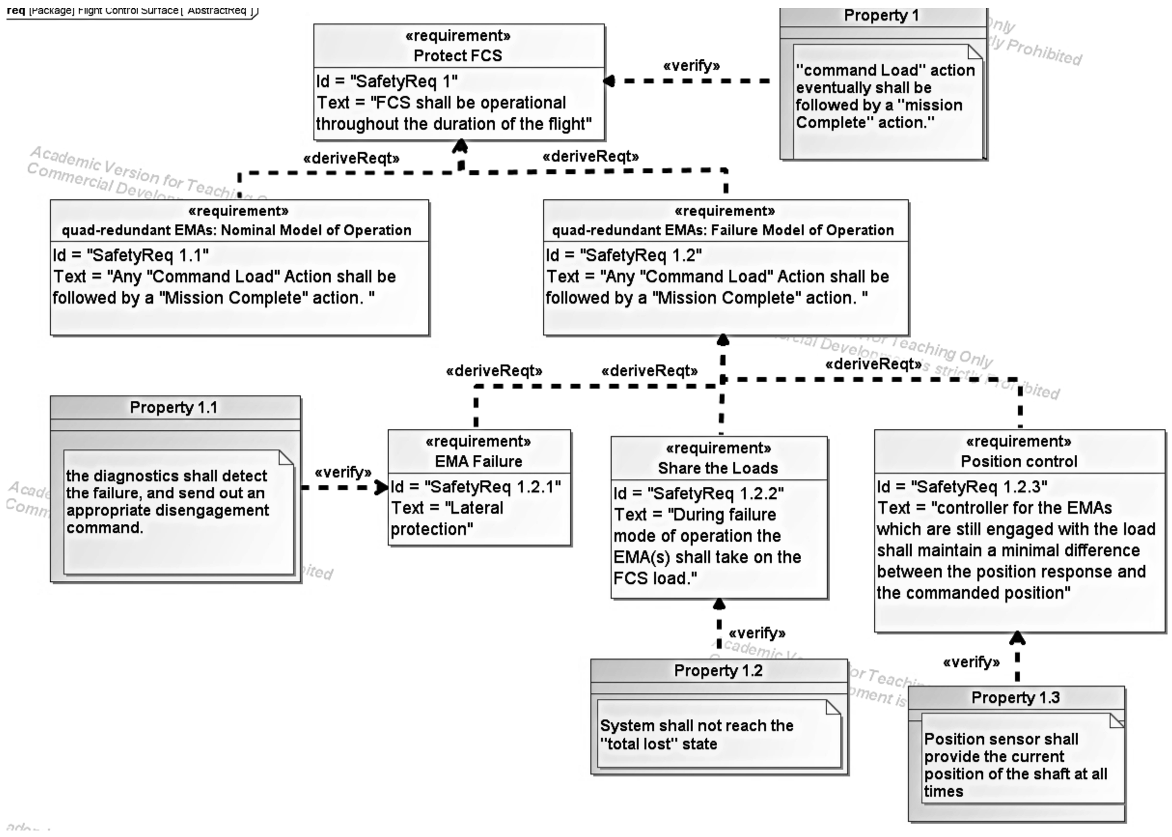

req [Package] Flight Control Surface [AbstractReq]
«requirement» Protect FCS
Id = "SafetyReq 1"
Text = "FCS shall be operational throughout the duration of the flight"
Property 1
"command Load" action eventually shall be followed by a "mission Complete" action.
«verify»
«deriveReqt»
«deriveReqt»
«requirement» quad-redundant EMAs: Nominal Model of Operation
Id = "SafetyReq 1.1"
Text = "Any "Command Load" Action shall be followed by a "Mission Complete" action. "
«requirement» quad-redundant EMAs: Failure Model of Operation
Id = "SafetyReq 1.2"
Text = "Any "Command Load" Action shall be followed by a "Mission Complete" action. "
«deriveReqt»
«deriveReqt»
«deriveReqt»
Property 1.1
the diagnostics shall detect the failure, and send out an appropriate disengagement command.
«verify»
«requirement» EMA Failure
Id = "SafetyReq 1.2.1"
Text = "Lateral protection"
«requirement» Share the Loads
Id = "SafetyReq 1.2.2"
Text = "During failure mode of operation the EMA(s) shall take on the FCS load."
«requirement» Position control
Id = "SafetyReq 1.2.3"
Text = "controller for the EMAs which are still engaged with the load shall maintain a minimal difference between the position response and the commanded position"
«verify»
Property 1.2
System shall not reach the "total lost" state
«verify»
Property 1.3
Position sensor shall provide the current position of the shaft at all times

Figure 3.9: Quad-redundant EMAs High-Level Requirements.

external load and the total actuator load response is used to accelerate or decelerate the output shaft. If the controller perceives that the output shaft position response is falling behind the commanded position, it will increase the current flow to the EMAs. As depicted in Table 3.3, in the FSP description of the controller, a repetitive behavior is defined using a recursion. In this context, recursion is recognized as a behavior of a process that is defined in terms of itself, in order to express repetition. The partial LTS model of the controller is depicted in Fig. 3.10. The *controller* performs action $<getLoad[l..4]>$, and then behaves as described by $<Controller[l]>$.

Table 3.2: Requirement Mapping.

Requirement	**Component(s)**
Safety Requirement 1	quad-redundant EMAs
Safety Requirement 1.2	quad-redundant EMAs
Safety Requirement 1.2.1	Diagnostics
Safety Requirement 1.2.2	EMAs
Safety Requirement 1.2.3	Controller, Position Sensor, and Shaft

Table 3.3: FSP Description of Controller

```
1: Controller = (getLoad[l:L] → Controller[l]),
2: Controller[t:L] = (timeout → Controller
3:     | sendLoad→allLoadsCompleted→getShaftPosition[x:Positions]
4:     →if (x ≥ t) then (missionComplete→Controller)
5:      else Controller[t]).
```

Controller[*l*] is a process whose behavior offers a choice, expressed by the choice operator "|". *Controller*[*l*] initially engages in either <*timeout*> or <*SendLoad*>. The action <*timeout*> is performed when all actuators fail, otherwise <*SendLoad*> is utilized. Subsequently, after sending the required load to each EMA, feedback signals are sent to inform the controller of completion of tasks by labeling the action with <*all Loads Completed*>. This results in the controller to perform the action <*get Shaft Position*>. At this stage, the controller compares the new position with the required shaft position, if the shaft has reached the required position then the <*mission is completed*>. Otherwise, the behavior is repeated until the shaft reaches the required position.

After modeling the behavior of each component and sub-system, the design is described by a *composition* expression. In the context of system design engineering, the term composition is similar to the coupled model. The coupled model defines

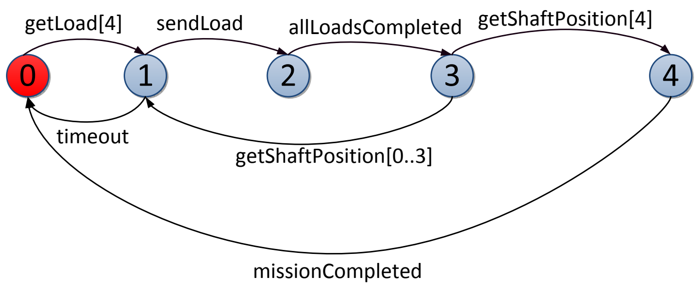


Figure 3.10: LTS Model of the Controller Subsystem.

how to couple several component models together to form a new model, similarly, composition groups together individual state machines. Such an expression is called a parallel composition, denoted by "‖". The "‖" is a binary operator that accepts two LTSs as an input argument. In the joint behavior of the two LTSs, the transition can be performed by any of the LTS if the action that labels the transition is not shared with the other LTS. Shared actions have to be performed concurrently. Table 3.4 depicts the FSP of the joint behavior of *EMA* and *controller*. The composed LTS model of the two subsystems consists of 161 states and 62 transitions. The shared action between the two models is the <*sendLoad*> action from the controller and the <*recLoad*> action from the EMA, therefore, these two are required to be performed synchronously. In order to change action labels of an LTS, the *relabeling* operator "/" is used, e.g., { recLoad / sendLoad }.

Table 3.4: Parallel Composition of EMA (Table 3.1) and Controller (Table 3.3)

1: ‖ **Leg = (EMA** ‖ **Controller) / { recLoad / sendLoad }.**

Table 3.5 presents some of the state transitions (or sequence of actions) produced by the composed model. Two possible executions under the EMA's nominal and faulty conditions are considered. In nominal mode, the EMA receives a request from a controller to provide two unit loads. At each time step, EMA performs one unit load and repeats until the output shaft reaches the required position that is when the <*missionComplete*> actions is performed. In the failed mode, initial actions are the same as in nominal mode until an EMA jams. The jammed EMA blocks the rest of the system from moving until it is disengaged. The process is followed by the <*Unblock*> action which unblocks the shaft allowing the rest of the system to be freed. By this time, the EMA has provided one unit load before being disconnected from the rest of the system. Since, the <$ShaftPositionIS$> shows the current position of the shaft being one instead of two, the EMA is required to perform one more unit of load. However, the disengaged EMA is incapable of doing so resulting in a <*timeout*>. The <*timeout*> occurs only when there are no EMAs to perform the required load.

Table 3.5: Leg Subsystem: Two Possible Transitions

EMA: Nominal Mode	EMA: Failure Mode
1: ctrl_getLoad.2	*1*: ctrl_getLoad.2
2: EMA_recLoad	*2*: EMA_recLoad
3: EMA_performLoad	*3*: EMA_performLoad
4: LoadsCompleted	*3*: **EMA_jam**
5: ShaftPositionIs.1	*4*: **Shaft_block**
6: EMA_recLoad	*5*: **EMA_Disengage**
7: EMA_performLoad	*6*: **Shaft_Unblock**
8: LoadsCompleted	*7*: LoadsCompleted
9: getShaftPosition.2	*8*: ShaftPositionIs.1
10: EMA_performLoad	*9*: **timeout**
11: missionComplete	–

So far, we provided the basis for decomposing and modeling the system based on the modular description of the design components and subsystems. In the next phase, the process of expressing the desired safety properties in terms of a state machine or LTS is described. The advantage is that both the design and its requirements are modeled in a syntactically uniform fashion. Therefore, the design can be compared to the requirements to determine whether its behavior conforms to that of the specifications. In the context of this work, the properties of a system are modeled as safety a FPS. A ***safety*** FPS contains no failure states. In modeling and reasoning about complex systems, it is more efficient to define safety properties by directly declaring the desired behavior of a system instead of stating the characteristics of a faulty behavior. In a FSP, the definition of properties is distinguished from those of subsystem and component behaviors with the keyword *property*.

Based on the requirement decomposition model of Fig. 3.9, the composition model of the properties *P1.1, P1.2, and P1.3* is presented by the following generic (or parameterized) safety property with the following constants and a range definitions is used:

- const N =4 \\ *number of faulty EMAs* [1]
- const M =4 \\ *number of EMAs*
- range EMAs = 1..M \\ *EMA identities*

In order to prevent the system from reaching the catastrophic event of <*timeout*>, it is essential to complete the mission and provide the required loads based on the command signal. The property of Table 3.6, maintains a count of faulty EMAs with the variable f. To model the fact that every command signal must be followed by a <*missioncomplete*>, property *P1*, the processes in lines 3 and 8 are required to

[1]by default is set to *4* but it can be redefined during the instantiation process.

constrain the number of faulty EMAs (f) to a number defined by the parameter of the property (*e.g. N=4*).

Table 3.6: FSP Model of Safety Property

```
1: property
2: Fault_Tolerance(N=4) = Jammed[0],
3: Jammed[f: 0..M] =(when(f ≤ N)commandLoad[L] → CompleteMission[f]
4:          |when (f>N) commandLoad[L] → Jammed[f]
5:          |d[EMAs].jam → Jammed[f+1]
6:          |missionComplete → Jammed[f]),
7: CompleteMission[f:0..M] = (missionComplete → Jammed[f]
8:          |when (f<N ) d[EMAs].jam → CompleteMission[f+1]
9:          |when (f= =N) d[EMAs].jam → Jammed[f+1]).
```

If the above property is predefined with $N = 2$, permitting only two out of four EMAs to fail during the system operation, the verification algorithm of Fig. 3.7 verifies that the safety property is satisfied. The composed LTS model consists of 2^{42} states, however the verification algorithm reduced the number of states to *10733*. The same result is obtained with three EMAs failing (*e.g. N=3*).

However, when the property is instantiated allowing four EMAs to fail, the safety analysis verifies that the property is violated and a failure propagation path is produced. Therefore, the generic safety property modeled in Table 3.6 verifies that the system never reaches the failure condition of *total loss* if and only if $N \leq$ *M-1* where N is the number of faulty EMAs and M is the total number of EMAs.

From the result of case study: the characterization of the system architecture by its subsystems and components improves requirements specification, tracking, and modeling. In addition, the FSP annotation of the failure behavior of each of component, and the system level safety analysis based on components' interaction lead to achieving a manageable verification procedure. As the compositional reasoning approach significantly reduces the number states to be explored, exhaustive checking of the entire state space is made feasible without the need for a exhaustive search. This is especially important where the exhaustive simulation is too expensive and non-exhaustive simulation can miss the critical safety violation.

3.5 Conclusion

There is a growing demand for formal methods and tools that facilitate the specification and verification of complex engineered systems design. Also, safety standards for the design of safety-critical systems strongly recommend the use of formal verification approach as part of the certification process. However, these standards do not specify how formal approaches can be implemented. Alternatively, system engineering semi-formal techniques for elicitation and structuring the requirements of complex engineered systems are essential part of the design for electing the conceptual design that satisfies the identified requirements.

In this paper, we have proposed a system modeling and verification approach that combines these apparently contradictory views. The semi-formal SysML techniques based on requirement and block diagrams combined with formal verification methods based on the assume-guarantee reasoning are used to prove that the behavior of sub-systems and components satisfies the design requirements. The proposed approach is based on the mapping between the hierarchical decomposition model of the requirements and properties to be satisfied, functions and behaviors to be realized, and sub-systems and components to be implemented.

The future work will continue in verifying more sophisticated system, while taking into consideration safety properties that are formulated using the temporal operators, i.e., until, before, or after. More complex temporal properties will be tested. In the case of temporal properties, satisfying the system property is not always equivalent to satisfying a local composition of sub-properties. The modified verification algorithm will use linear temporal logic (LTL) as a specification formalism.

Chapter 4: Verification Based on Assume-Guarantee Reasoning

Hoda Mehrpouyan, Irem Y. Tumer, Chris Hoyle, Dimitra Giannakopoulou, Guillaume Brat

Journal of Systems Engineering

Submitted Jun 2014

4.1 Introduction

With increasing complexity in the design of complex engineered systems such as aerospace, maritime, nuclear, and major civil infrastructure systems, the cost and time required for design and development are growing at an unsustainable rate. For instance, Boeing and Airbus experienced significant delays and cost overruns in delivering their latest 787 Dreamliner jet and A380 superjumbo projects [1]. The underlying causes were cited as issues in the A380 design and production, which resulted in a variety of glitches such as engine blow-up, failure of the backup brakes, and discovery of cracks on the wings of the planes. Boeing 787 Dreamliner faced similar issues, as described in the national transportation safety board reports of the fire incident in the auxiliary power unit (APU) on a Japan Airlines 787 flight from Boston on Jan. 2013. The report concluded that a design flaw might be the root cause of the fire. In another case, a Japan Airline Dreamliner flight from Boston faced a considerable delay after a fuel leak resulting from a faulty valve. The design flaws and equipment malfunction cost \$5 Billion for Boeing on top of the \$1 Billion compensation claim from airlines, such as United Airlines and Air India.

Therefore, in order to improve the design and development process, manufacturing companies are increasingly relying on simulations to understand the unexpected behavior of the design to improve both robustness and performance of system. [2], and to enable the Verification and Validation (V&V) of system design. V&V are techniques used for confirming that a design satisfies its requirements and performs its intended functions [3]. Therefore, system verification and validation are critical to reduce design faults and facilitate effective and efficient design changes.

Plant et al. [4] define verification as the process of evaluating whether or not a design complies with requirement specifications. On the other hand, validation is defined as the process of identifying whether the mathematical models sufficiently represent the reality with regard to decisions that have to be made during the design and development process. These decisions consist of requirement validation and design validation. The goal of requirement validation is to confirm that the system requirements are sufficiently correct, consistent, and complete to achieve reliability and safety to meet the needs of stakeholders within the design and development constraints, i.e., schedule and cost.

Based on his research, Foster [5] concluded that verification process consumes over 60% of the design time. With the magnitude of modern design spaces combined with increasing design complexity, the verification process takes even longer. In addition to longer verification time which results in project cost increase, a survey conducted by Collett International Research Inc. [6] revealed that the traditional verification approaches do not cover all possible behaviors of a system; that is, they are not able to find all violations of a system specification. Therefore, this paper focuses exclusively on the design verification to provide proof that the design satisfies the safety requirements. The main functionalities provided by the proposed approach include automatic failure injection based on a database of predefined failure modes, automatic generation of fault trees, and exhaustive safety property verification with the help of model checking algorithms.

The remainder of this paper is structured as follows: Section 2 presents the background and related research on failure analysis techniques in the early stages of system design, while discussing their strengths and weaknesses. In addition, the definition of *assume-guarantee reasoning* and its commonly used terminologies and operators are addressed in Section 2. In Section 3 an overview of the step-by-step implementation of the assume-guarantee reasoning algorithm on the components of the design architectures is explained. Section 4 outlines the application of the proposed methodology in the analysis and verification of the safety properties of the satellite electrical power system design. The paper ends with conclusions and future work.

4.2 Background

A variety of modeling approaches and tools are used in industry or in academia, e.g., AADL [7], Modelica [8], Ptolemy [9], MATLAB/Simulink [10], SysML [11]. These tools are used to model the functionality and architecture of the system design, then simulation is carried out to verify the design. However, most of the simulation experiments are designed to evaluate a limited set of scenarios in order to deal with the system complexity. The effects of this informal and incomplete verification is the possibility that a non-tested scenario could result in unexpected behavior and catastrophic system failure. To address the incomplete verification of designs via simulation, formal methods have been proposed to increase the confidence level. Formal verification enables the evaluation of safety properties at different levels of abstractions,(i.e., component, sub-system, system), guaranteeing the systems' behavior in every possible scenario.

The main objective of the verification process is to make sure that the design complies with safety requirements. In order to satisfy most regulatory guidelines and safety standards, designers must develop a safety case to prove the safety justification of a design. These cases should represent *all potential* hazards and appropriate steps be taken to rectify the situation. These types of safety documents usually include safety specifications, results of failure and risk analysis, verification approach, and results of all the verification activities. Figure 4.1 depicts the general view of the verification process.

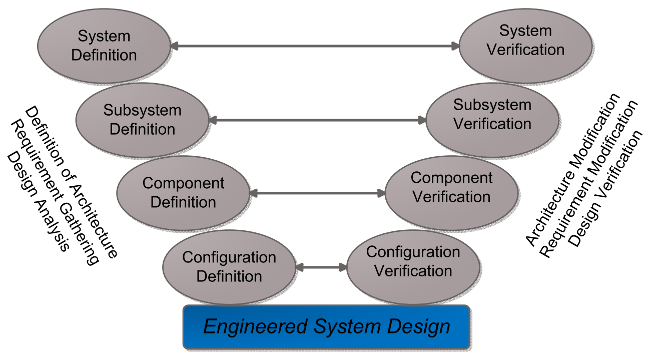


Figure 4.1: Verification Process In Engineered System Design.

4.2.1 Verification Based On Formal Methods

In [12], Henzinger et al. cover the advantages that formal verification offers over the above approaches. In formal verification, system designers construct a precise mathematical model of the system under design, so that extensive analysis is carried out to generate proof of correctness. One of the well-established methods for automatic formal verification of the system is model checking, where a mathematical model of a system is constructed and verified with regards to specified properties. In model checking, the desired properties are defined in terms of temporal logic [13]. The defined logical formulae are then used to prove that a system design meets safety requirements and specifications. A model checker to establish assume-guarantee properties of components is called assume-guarantee reasoning (AGR) [14, 15, 16, 17]. In the assume-guarantee reasoning (AGR) method the system properties are verified and modeled with respect to the *assumptions* on the environment where component and (sub)system performances are *guaranteed* under these assumptions. The assumption generation methodology uses compositional and hierarchical reasoning approaches via a compositional reachability analysis (CRA) [18] technique. CRA incrementally composes and abstracts the component models into subsystem and, ultimately, a high-level system models. Based on the assume-guarantee reasoning (AGR) paradigm, *assume-guarantee* can be defined as a pair of assumptions and guarantees which formally describe:

1. The context in which the system design is *assumed* to be used.

2. The requirements which the system design demands to *guarantee* correct operation. (It is important to note that "guaranteeing the correct operation in an assumed environment" is only possible with a specified probability. In this context "guarantee" does not mean that system will always survive the assumed environment without any failures, it means that system will survive at the probability level in which one specifies, i.e, .9999).

Assume-guarantee reasoning has been widely used in the computer science literature as a means for software verification. As a mathematical foundation for representing the engineering requirements, this approach can be used for verification of complex engineered system design. Additionally, the work focuses on safety property specification and design verification at multiple abstraction layers.

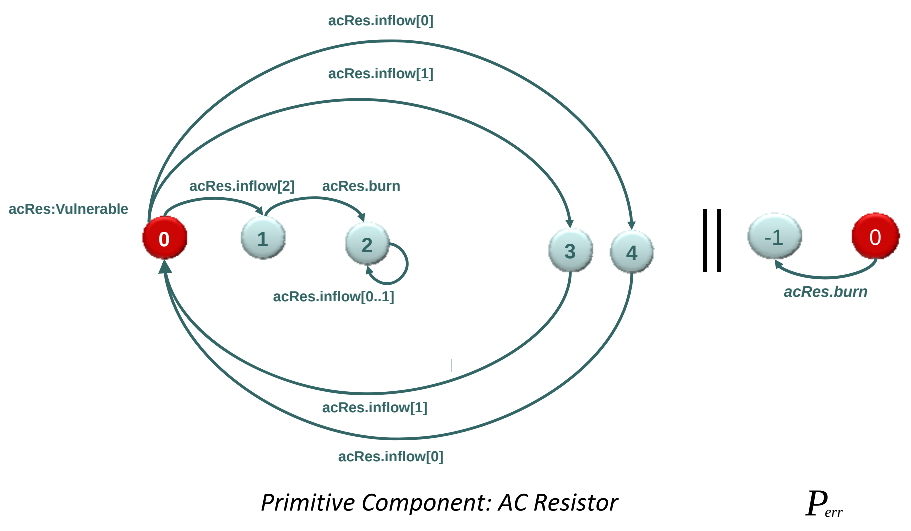
As discussed in the previous section, abstraction and composition are the two most used principles in any system verification methodology for handling the complexity and analysis of engineered systems. When verifying complex systems, different approaches (e.g., model-based methods) use hierarchical abstraction layers such as functional, structural, and behavioral models to represent the system under study. A functional model is a representation of all the necessary functions that the system must contain in order to meet the design requirements. Kurtoglu et al. [19, 20] present a framework for developing a functional model for the hardware components, while Wang et al. [21] suggest object oriented programming for modularization and functional modeling of the software components. Functional models provide the required information about the flow of EMS and data between components throughout the system design. In functional modeling [22, 23], the EMS flows and functions are modeled using nouns and verbs respectively, e.g., store electricity, actuate electricity, etc. The functional model of the design is developed based on the hierarchical structure of functions and flows. Next, the structural model as a suitable design solution is developed. The structural model describes different system components and the EMS flow relationship between them. Using different design solutions within the complex system design process, various design concepts for a system are developed. While all design concepts share the same functional description, they are implemented differently. They are different in structure and behavior. Finally, the behavioral model of a component contains the nominal and failure states of the component, including transitions leading to these states. The behavioral model results from the relationship between input/output flows and the underlying first principles. Once the behavioral models of the component are developed, they are incorporated into the Labeled Transition Systems (LTSs) model by mapping them with their respective LTSs transitions.

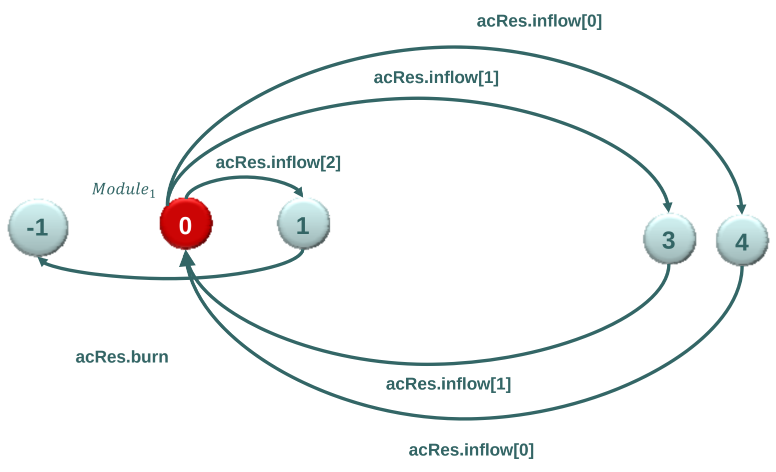
4.3 Methodology

The contribution of this paper is developing an automated design verification framework to prove the correctness of the complex engineered system design with regards to its functional and safety properties. The proposed framework provides information on the property violation of the composed components during conceptual design, while identifying the failure propagation behavior. The automatic generation of failure propagation paths enables the system designers to better address the safety issues in the design.

4.3.1 System Modeling

In the proposed approach, finite-state model of a system is analyzed to ensure satisfaction of safety properties that assure a desired system behavior. Finite-state model is a representation of the system behavior that is generated in the form of Finite State Process (FSP) [24]. FSP is an algebraic notation that is used to describe the component's behavior (Table 4.1). System designers create the FSP models which designed to be machine readable, and thus provides an ideal language to specify abstract model of the component's behavior or function. The developed FSP is then used through a modeling tool such as the Labelled Transition System Analyzer (LTSA) [25] to





|| $Module_1$ = (acRes:(Vulnerable || P_{err})).

Figure 4.2: Parallel Composition of Primitive Component and its Safety Property in LTS Format

provide compilation of FSPs into a Labelled Transition System (LTS) [26, 25]. The LTS model is expressed graphically by its alphabet, transition relation, and states including single initial state (Fig. 4.2). The LTS of the system is constructed from the LTS of its subsystems, and is verified against safety properties of the design requirements. In this research, the model checking algorithm is integrated as part of the LTSA tool which performs exhaustive execution of all system's behavior to determine if the safety property is violated or not. The LTSA takes advantage of the method called compositional reachability analysis (CRA) and creates a reachability graph for the system which contains information about the safety values of each state. A safety property then is checked by analyzing the reachability graph, searching for paths on which the safety property is violated. Labelled Transition Systems (LTS) $\boldsymbol{T}$ is defined as:

A set **S** of **states**

A set **L** of **actions**

A set $\rightarrow$ of transitions from one state to another.

An initial state $s_0 \in S$

$\mathbf{T} = (S, L, \rightarrow, s_0)$.

4.3.2 Parallel Composition of LTSs

In this section the parallel composition operator [27] and the composition mechanism that assists with compositional modeling of the component models are explained.

The parallel composition operator enables both associative and commutative composition; therefore the order of LTSs models that are composed together is insignificant. The parallel composition operator, denoted by "||", is a binary operator that accepts two LTSs as an input argument. Based on the definition of this operator, composed LTSs interact by synchronizing on common actions (i.e., exchange of EMS) shared in their FSP models with interleaving of the remaining actions. Designing interacting components with LTSs is therefore sensitive to the selection of action names. In addition, parallel composition is based on the model instantiation which is defined by constructing a copy of a LTS model where each transition label is prefixed by the name of the instance.

4.3.3 Electrical Power System Design

Validation of the proposed verification framework is through application to an Electrical Power System (EPS) which is designed to provide power to selected loads. In an aerospace vehicle these loads usually include subsystems such as the propulsion, avionics, and thermal management systems. The basic functionality that EPS is required to provide is common to many aerospace applications such as power storage, power distribution, and operation of loads [28]. Fig. 4.3 displays the existing design of the EPS, containing a power source connected through a series of relays to an inverter and several loads consisting of a large fan, a Direct Current (DC) resistor and a Alternating Current (AC) resistor. In order to create an integrated health management environment a sensor suite is designed to enable monitoring of currents, voltages, and temperatures through out the circuit. A series of four AC or

DC voltage sensors and three current transmitters measure the voltage and current at different points throughout the circuit. Fig. 4.5 depicts a functional model of the EPS, which serves as the baseline architecture for this research. Fig. 4.3 displays the Modelica [29] representation of the EPS design.

In the EPS test-bed, the power source component that is denoted by a circle may have the operational modes on and off. The on-mode has the functional action of generating power, which can also result in over-current spikes. As illustrated in Fig. 4.3, the generated spike affects the AC resistor, fan, and DC resistor that are denoted by circles on the right hand side of the figure. These components are vulnerable to the spike generated by the power source. Thus, safety properties are needed to protect these vulnerable components and ensure the proper operation of the whole system. The safety properties define the types of failure that a component is vulnerable to and must be checked to ensure the failure state is not reached. As depicted in Fig. 4.3, there are three different paths *1.* $\{ A, B, C \}$, *2.* $\{ A, B, D \}$, *and 3.* $\{ A, E \}$ in which the generated spike from the power source can reach the three vulnerable components; these are considered design flaws [30].

4.3.4 Reusable Models and Binding Interfaces

In the proposed system modeling approach, each component defines a scope for the transition in its behavioral model (e.g., an instance of the resistor is defined by the use of the command *res:resistor*). Instantiation permits the ***reuse*** of LTSs during system modeling through multiple instantiations, e.g., Alternating Current (AC) and Direct Current (DC) resistors behave in a similar fashion under the influence when

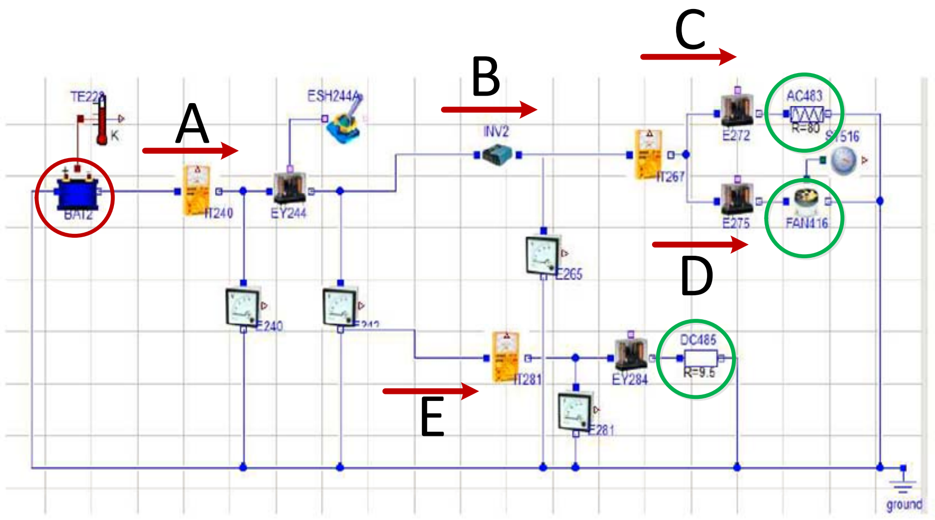


Figure 4.3: Structural Model of the EPS

the power received from the environment exceeds the resistor's ability to dissipate the heat and therefore the resistor can be used for both components.) Instantiation creates unique labeling of transitions in the LTS models (e.g., each transition in the AC resistor's behavioral model is labeled with a prefix of ***"ACres."*** and each transition in the DC resistor's model is labeled with a prefix of ***"DCres."***

In order to create the compositional model of the circuit breaker and AC resistor, an instance of the circuit breaker is defined by the use of the command *cb:CircuitBreaker* that is composed with the previously defined instance *ACres*. However, the two LTSs do not have any alphabet in common so no synchronization is possible. For this reason, the binding between the two models is created by the use of the command *ACres.inflow/cb.outflow*. The binding leads to a model where the circuit breaker's output current is recognized as the AC resistor's input current. As a result, the new label *ACres.inflow*, is substituted with the old label, *cb.outflow*. With the binding command in place, the synchronization takes place and the LTSs components have a common action to communicate.

In our context, properties are modeled as safety LTSs. A ***safety*** LTS is a LTS that contains no failure states. When checking a property P, an error LTS denoted P_{err} is created, which identifies possible violations with the failure state.

4.3.5 Verification Process

The contribution of this paper is developing an automated design verification framework to prove the correctness of the complex engineered system design with regards to its functional and safety properties. The proposed framework provides information on the property violation of the composed components during conceptual design, while identifying the failure propagation behavior. The automatic generation of failure propagation paths enables the system designers to better address the safety issues in the design. Fig. 4.4 depicts the relationship between system models and the verification framework that either provides the proof of correctness or failure propagation information.

The main steps to apply the assume-guarantee reasoning (AGR) framework are summarized as follows:

1. A functional model is generated as a representation of all the necessary functions that the system must contain in order to meet the design requirements is used to create the structural model of the system.

2. The structural model is used as a resource to gather information about the different system components and the Energy, Material, and Signal (EMS) flow

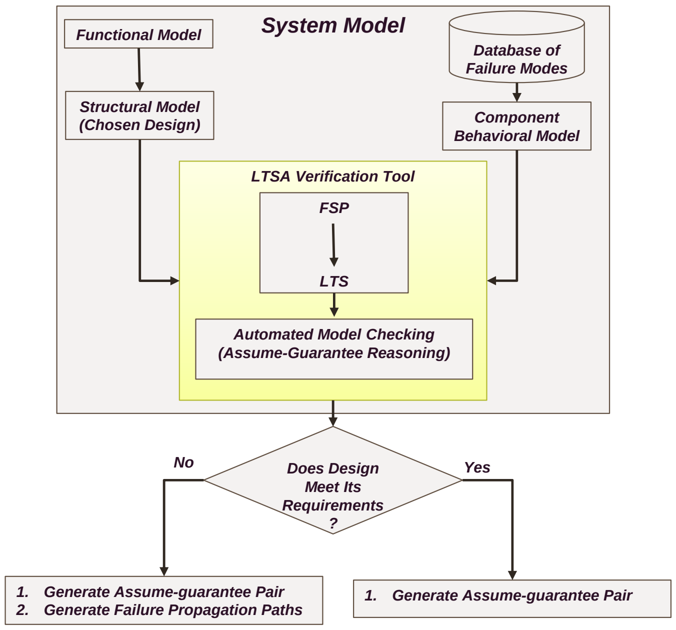

System Model
Functional Model
Structural Model (Chosen Design)
Database of Failure Modes
Component Behavioral Model
LTSA Verification Tool
FSP
LTS
Automated Model Checking (Assume-Guarantee Reasoning)
Does Design Meet Its Requirements ?
No
Yes
1. Generate Assume-guarantee Pair
2. Generate Failure Propagation Paths
1. Generate Assume-guarantee Pair

Figure 4.4: An Overview of the Proposed Verification Process.

relationship between them to create a database of their failure states and safety properties and the behavioral models.

3. Once the behavioral models of the component are developed, they are incorporated into the Finite State Processes (FSP) by mapping the behavior models with their respective FSPs' transitions.

4. The Labeled Transition System (LTS) of the system is automatically constructed from the FSP models that are developed by design engineers.

5. The LTSs are incrementally analyzed and abstracted through the use of a reachability graph to determine whether the safety property is violated.

6. If the failure state of the component is reached, then the failure propagation path is provided.

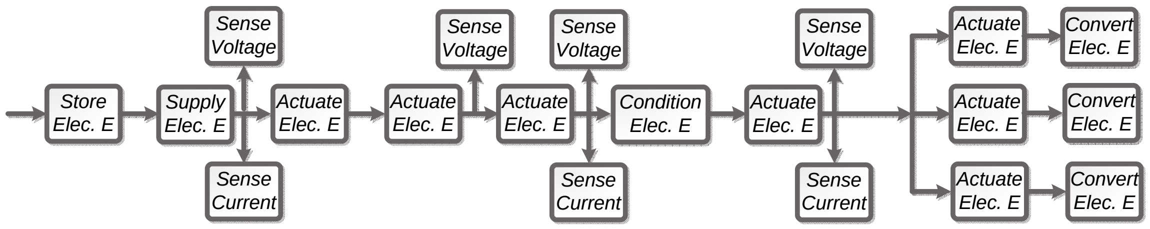


Figure 4.5: Functional Model of the Electrical Power System.

The AGR paradigm requires exact identification of the component properties, which in this case are defined based on the failed states, of the components and (sub)systems. In order to identify the failed states the effects of incoming EMS flows on the operation of the components is analyzed and two generic states of nominal and failed are defined. The component's state is recognized as ***nominal*** when a component is operating with the performance and functionality intended by the system designer. On the other hand, ***failure*** state is defined as a component functioning in a way that was not intended by the designer.

4.3.6 The LTSA Tool

The Labeled Transition System Analyzer (LTSA) [31, 25] is an automated tool that supports Compositional Reachability Analysis (CRA) [18] of a software system based on its architecture. In general, the conceptual design of a complex engineered system has a hierarchical structure and is modular [32]. CRA incrementally computes and abstracts the behavior of composite components based on the behavior of their immediate children in the hierarchy. The input language "FSP" of the tool is a process-algebra style notation with Labeled Transition Systems (LTS) semantics.

A property is also expressed as an LTS, but with extended semantics, and is treated as an ordinary component during composition. Properties are combined with the components to which they refer. They do not interfere with system behavior unless they are violated. In the presence of violations, the properties introduced may reduce the state space of the (sub)systems analyzed. As in our approach, the LTSA framework treats components as open systems that may only satisfy some requirements in specific contexts. By composing components with their properties it postpones analysis until the system is closed, meaning that all contextual behavior that is applicable has been provided.

4.4 Case Study

Continuing with the EPS design concept, the development of LTSs implies direct mapping between the functional model (Fig. 4.5) and the structural architecture (Fig. 4.3) of the system, in which specific component or (sub)system is selected to implement the functional requirements in the actual system design. For example, in Fig. 4.3 the component "inverter" implements function "condition electricity" of Fig. 4.5.

In order to construct the LTS model of the EPS design, all internal state transitions of the components are presented in the Finite State Processes (FSP) language. The constant variables(factors that do not change during the course of this experiment) and ranges are defined as follows:

const Low = 0

const Medium = 1

const Spike = 2

const Open = 1

const Close = 0

range CUR = Low..Spike

The notations 0,1, and 2 are used to denote low, medium, and spike currents in the EPS components, respectively.

4.4.1 Primitive Components and Properties

As depicted in Table 4.1 all internal state transitions of the primitive components are presented in the FSP language. Each component's nominal behavioral model is incorporated into the FSP code, therefore the resulting model contains no failure state. For example, while the AC resistor is in the operational mode (Table 4.1), two transitions are possible, which are specified using the "OR" logical operator "|" . These two transitions are defined as: 1- the current input into the AC resistor which is in the range of low or medium and the resulting output current also in the low or medium range 2- the current in-flow to the AC resistor is spiking which results in the state of "burn". If the resistor is in the state of burn, no matter what input current level to the AC resistor, there is no output current past the AC resistor.
The failure modes of components are represented by P_{err}. The error LTSs are constructed to represent all the faulty transitions that lead to failure states. In order to model the failure mode of the three vulnerable components discussed in design architecture of Fig. 4.3, a generic property named P_{err} is defined for all three components as below:

property P_{err} = STOP + burn.

Table 4.1: EPS Components and their FSP Models

EPS System		
Component	Mode	FSP Model
Battery	nominal	(inflow[v:CUR] → outflow[v] → Battery)
Current Sensor	nominal	(inflow[v:CUR] → outflow[v] → Current Sensor)
Voltage Sensor	nominal	(inflow[v:CUR] → outflow[v] → Voltage Sensor)
Relay	nominal	(inflow[v:CUR] → if (Open) then (outflow[v] → Relay else Relay))
Inverter	nominal	(inflow[v:CUR] → outflow[v] → Inverter)
AC Resistor	nominal failure	Operational = (inflow[v:CUR] → outflow[v] → Operational | inflow[Spike] → burn → BURNED), Burned = (inflow[CUR] → Burned)
DC Resistor	nominal failure	same as AC Resistor
Fan	nominal failure	same as AC Resistor

The LTSA tool represents failure state by ***-1*** as it is depicted in Fig. 4.2 in the compositional model of the AC resistor with its property that is reached by the illegal transition of *acres.burn*.

4.4.2 Compositional Model

In order to create the compositional model of the EPS system, the order of compositions is decided based on the functional model of the design (Fig. 4.5). Table 4.2 represents the compositional model of the EPS system for two types of components. Those components that operate in nominal mode such as "battery" and "$currentsensor_{240}$" in $module_{18}$ are composed by the creation of binding between them. The binding is modeled by the EMS flow between the two components which is represented by the use of the command $cs_{240}.inflow/bat_2.outflow$. The binding leads to the compositional model where the battery's output flow is recognized as $currentsensor_{240}$'s input flow. The second types of components are those with failure states, which are required to be composed with their defined properties before

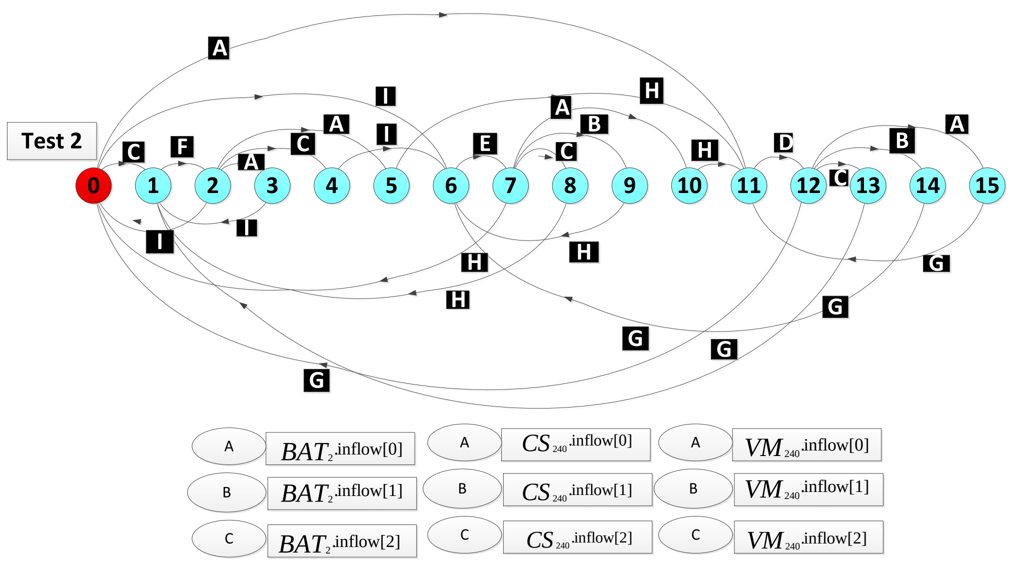


Figure 4.6: Parallel Composition in LTS Format

they can be considered for composition with other components, e.g., *module*$_1$ *through module*$_3$ in Table 4.2.

Also it is important to note that the FSP language supports *component instantiation*, which is defined by constructing a copy of a model where each transition label is prefixed by the name of the instance. As a result, each component defines a scope for the transition in its behavioral model. For example, in the compositional modeling of the EPS system, the instances of the *vulnerable* model are copied and reused for the AC resistor, DC resistor, and Fan, because of their similar behavior with regard to over current spike. Instantiation creates unique labeling of transitions in the LTSs. As illustrated in Fig. 4.2, each transition in the AC resistor model is labeled with a prefix of *"acres.###"*. Once the primitive components, their properties, and the compositional models are created, the LTSA tool is used to compile the models from FSP code into the LTS models (Fig. 4.2).

Table 4.2: Composition of the EPS components

EPS System Compositional Model
$\|$ $Module_1$ = (acRes:(Vulnerable $\|$ P_{err})).
$\|$ $Module_2$ = (fan:(Vulnerable $\|$ P_{err})).
$\|$ $Module_3$ = (dcRes:(Vulnerable $\|$ P_{err})).
$\|$ $Module_4$ = (rel272:Relay) / acRes.inflow/rel272.outflow.
$\|$ $Module_5$ = (rel275:Relay) / fan.inflow/rel275.outflow.
$\|$ $Module_6$ = (cs267:CurrentSensor) / rel272.inflow/cs267.outflow.
$\|$ $Module_7$ = (cs267:CurrentSensor) / rel275.inflow/cs267.outflow.
$\|$ $Module_8$ = (vm256:VoltMeter) / cs267.inflow/vm256.outflow.
$\|$ $Module_9$ = (inv2:Inverter) / vm256.inflow/inv2.outflow.
$\|$ $Module_{10}$ = (rel284:Relay) / dcRes.inflow/rel284.outflow.
$\|$ $Module_{11}$ = (vm281:VoltMeter) / rel284.inflow/vm281.outflow.
$\|$ $Module_{12}$ = (cs281:CurrentSensor) / vm281.inflow/cs281.outflow.
$\|$ $Module_{13}$ = (vm242:VoltMeter) / inv2.inflow/vm242.outflow.
$\|$ $Module_{14}$ = (vm242:VoltMeter) / cs281.inflow/vm242.outflow.
$\|$ $Module_{15}$ = (rel244:Relay) / vm242.inflow/rel244.outflow.
$\|$ $Module_{16}$ = (vm240:VoltMeter) / rel244.inflow/vm240.outflow.
$\|$ $Module_{17}$ = (cs240:CurrentSensor) / vm240.inflow/cs240.outflow.
$\|$ $Module_{18}$ = (bat2:Battery) / cs240.inflow/bat2.outflow.

4.4.3 Compositional Verification

In order to verify the properties of the EPS system, the LTSA "compositional" algorithm is used. This algorithm implements assume-guarantee reasoning in a learning framework to prove that the properties are satisfied or violated. The advantage of using model checking and automata learning algorithm is its ability to perform CRA in an exhaustive manner to search for violations of design properties. In addition, the LTSA algorithm uses a specific form of learning algorithm based on minimization and abstraction, which dramatically reduces the number of state spaces required for analysis. For example, if the two modules of the EPS LTSs, e.g, $module_{18}$ and $module_{17}$ are analyzed in a monolithic manner ($\|$ Test = ($Module_{18} \| Module_{17}$) the state space of this composition results in 16 states with 27 transitions as illustrated in Fig. 4.6. Eventually, the full monolithic composition of the EPS design results in approximately 232×10^9 states, however, with the proposed method in this paper, the compositional analysis is completed in 2 seconds.

The result of EPS compositional verification concluded by the AGR for the design of Fig. 4.3 was that the "***system and environment are incompatible***". The reason for this conclusion is that the EPS design represented in Fig. 4.3 assumes normal operating condition for the system. In normal condition, all three susceptible components receive nominal voltage and current, while any variation in load and distribution has an effect on the system. Therefore, the analyzed design is not considered *fault tolerant*.

In addition to verifying the desired properties of the system design, the proposed methodology automatically computes the required assume-guarantee pair for each component in the design to prove the global properties of the design under consideration. There are cases where no assume-guarantee pair is generated by the verification algorithm because there is no environment in which the design can be implemented safely. Fig. 4.7 represents the assume-guarantee pairs generated by the AGR for each system element in the design, which implies that each component guarantees to output current flow of low or medium (0 or 1) *iff* they receive current in-flow of low or medium. In the case of detecting safety violation in the system design, the verification framework returns a counterexample, which provides information of the failure propagation path. Only one counterexample is necessary to prove that the design violates its properties. In the case of EPS design, the failure propagation path starts from "battery" propagating through different components such as $CurrentSensor_{240}$, $VoltMeter_{240}$, $Relay_{244}$, $VoltMeter_{242}$, $Inverter_{2}$, $VoltMeter_{256}$, $CurrentSensor_{267}$, $Relay_{272}$ and reaching the "AC resistor" causing it to burn. The "*inflow.2*" represents the existence of the spike in the incoming and outgoing current flow in each component.

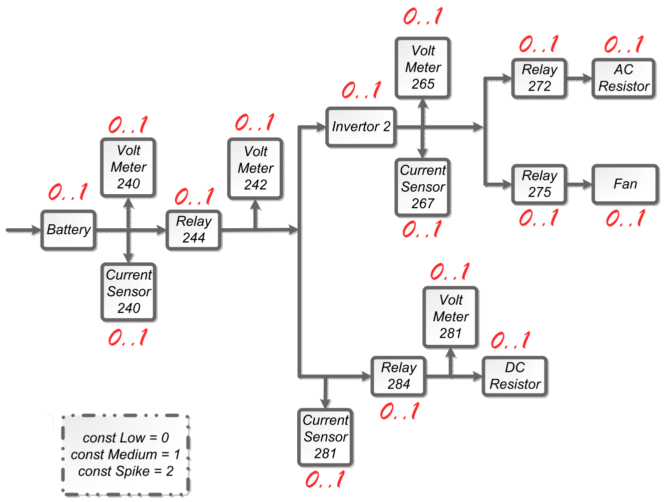


Figure 4.7: Assume-Guarantee Pair for the EPS Design of Fig. 4.3.

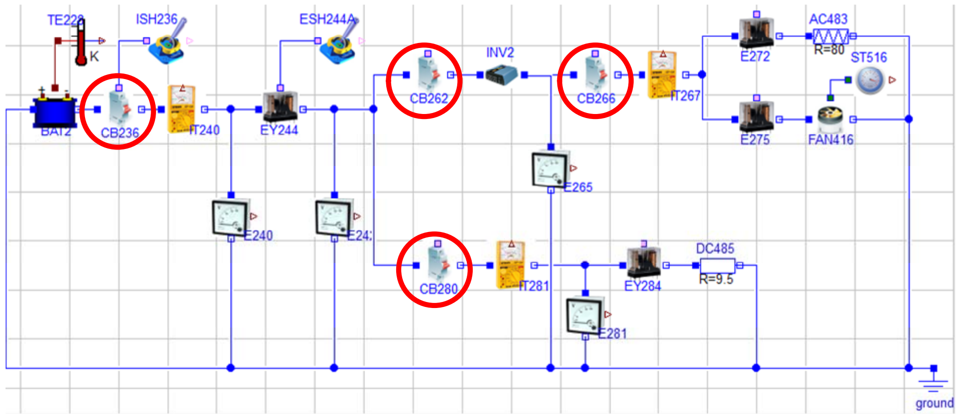


Figure 4.8: Model of the EPS with Circuit Breaker.

4.4.4 Design Based On The Result Of Verification

In order to correct the design flaws mentioned above, it is required to add circuit breakers to the design as modeled in Fig. 4.8 to prevent the spike reaching the three vulnerable components. The circles highlight the circuit breakers used to protect the AC resistor, fan, and DC resistor from an over-current spike. The operation of the circuit breaker is similar to that of an electrical switch, which is designed to protect an electrical circuit from damage caused by overload or short circuit (Table 4.3). Therefore, the integration of circuit breakers in the design architecture of Fig. 4.8 prevents the resistors and fan from burning. The following equation represents the assume-guarantee reasoning rule of triple type:

1. <{0..1}> *ACResistor* <No Burn>

2. <*true*> *CircuitBreaker* <{0..1}>

<*true*> AC Resistor || Circuit Breaker <No Burn>

(1) <***{0..1}***> ***AC resistor*** <***No Burn***> ***is*** proven correct if *circuit breaker* satisfies the assumption that in-flow current to the AC resistor is always {0..1}, resulting in guaranteeing property *No Burn*.

The compositional model of the modified design is represented in Table 4.4. The

Table 4.3: FSP Code for Circuit Breaker

Circuit Breaker		
Component	Mode	LTS Model
Circuit Breaker	nominal	(inflow[v:CUR] → if (v < Spike) then (outflow[v] → CircuitBreaker) else CircuitBreaker) + {outflow[Spike]}

full monolithic composition of the modified EPS design results in 19×10^{12} states, however, with the proposed method in this paper, the compositional analysis is completed in 2.5 seconds. The result of verification is successful, implying that the "system and environment are compatible" and all the design safety requirements are met.

Table 4.4: Composition of the EPS components

EPS System Compositional Model
‖ $Module_1$ = (acRes:(Vulnerable ‖ P_{err})).
‖ $Module_2$ = (fan:(Vulnerable ‖ P_{err})).
‖ $Module_3$ = (dcRes:(Vulnerable ‖ P_{err})).
‖ $Module_4$ = (rel272:Relay) / acRes.inflow/rel272.outflow.
‖ $Module_5$ = (rel275:Relay) / fan.inflow/rel275.outflow.
‖ $Module_6$ = (cs267:CurrentSensor) / rel272.inflow/cs267.outflow.
‖ $Module_7$ = (cs267:CurrentSensor) / rel275.inflow/cs267.outflow.
‖ $Module_8$ = (cb266:CircuitBreaker) / cs267.inflow/cb266.outflow.
‖ $Module_9$ = (vm256:VoltMeter) / cb266.inflow/vm256.outflow.
‖ $Module_{10}$ = (inv2:Inverter) / vm256.inflow/inv2.outflow.
‖ $Module_{11}$ = (cb262:CircuitBreaker) / inv2.inflow/cb262.outflow.
‖ $Module_{12}$ = (rel284:Relay) / dcRes.inflow/rel284.outflow.
‖ $Module_{13}$ = (vm281:VoltMeter) / rel284.inflow/vm281.outflow.
‖ $Module_{14}$ = (cs281:CurrentSensor) / vm281.inflow/cs281.outflow.
‖ $Module_{15}$ = (cb280:CircuitBreaker) / cs281.inflow/cb280.outflow.
‖ $Module_{16}$ = (vm242:VoltMeter) / cb262.inflow/vm242.outflow.
‖ $Module_{17}$ = (vm242:VoltMeter) / cb280.inflow/vm242.outflow.
‖ $Module_{18}$ = (rel244:Relay) / vm242.inflow/rel244.outflow.
‖ $Module_{19}$ = (vm240:VoltMeter) / rel244.inflow/vm240.outflow.
‖ $Module_{20}$ = (cs240:CurrentSensor) / vm240.inflow/cs240.outflow.
‖ $Module_{21}$ = (cb236:CircuitBreaker) / cs240.inflow/cb236.outflow.
‖ $Module_{22}$ = (bat2:Battery) / cb236.inflow/bat2.outflow.

It is important to note that the generated assume-guarantee pairs, as depicted in Table 4.5, restrict the current-inflow of low and medium, $[0, 1]$, for a smaller number of components, compared to the previous design which was analyzed in Fig. 4.7. The reason for this is that the $circuitbreaker_{266}$ detects a fault condition and interrupt current flow from reaching the two vulnerable components (AC resistor and fan) in the top branch, while the $circuitebreaker_{280}$ protects the lower branch. Therefore, any components before these two circuit breakers can accept the in-flow current of low, medium, and spike. This is a good indication of the weakest assumptions generated by the proposed framework, guiding the designers in their understanding of the design requirements. Based on the generated assumptions the two circuit breakers (236 and 262) are not required and therefore can be eliminated from the design.

Table 4.5: Generated Assume-Guarantee Pair of Fig. 4.8

For EPS Design of Fig. 4.8		
Component	Assumption	Guarantee
Battery	$[0, 1, 2]$	$[0, 1, 2]$
$Circuit\ Breaker_{236}$	$[0, 1, 2]$	$[0, 1]$
$Volt\ Meter_{240}$	$[0, 1, 2]$	$[0, 1, 2]$
$Current\ Sensor_{240}$	$[0, 1, 2]$	$[0, 1, 2]$
$Relay_{244}$	$[0, 1, 2]$	$[0, 1, 2]$
$Volt\ Meter_{242}$	$[0, 1, 2]$	$[0, 1, 2]$
$Circuit\ Breaker_{262}$	$[0, 1, 2]$	$[0, 1]$
$Invertor_{2}$	$[0, 1, 2]$	$[0, 1, 2]$
$Circuit\ Breaker_{266}$	$[0, 1, 2]$	$[0, 1]$
$Volt\ Meter_{265}$	$[0, 1]$	$[0, 1]$
$Current\ Sensor_{267}$	$[0, 1]$	$[0, 1]$
$Relay_{272}$	$[0, 1]$	$[0, 1]$
$Relay_{275}$	$[0, 1]$	$[0, 1]$
$ACResistor$	$[0, 1]$	$[0, 1]$
Fan	$[0, 1]$	$[0, 1]$
$Circuit\ Breaker_{280}$	$[0, 1, 2]$	$[0, 1]$
$Current\ Sensor_{281}$	$[0, 1]$	$[0, 1]$
$Relay_{284}$	$[0, 1]$	$[0, 1]$
$Volt\ Meter_{281}$	$[0, 1]$	$[0, 1]$
$DCResistor$	$[0, 1]$	$[0, 1]$

The second alternative design architecture of the EPS system with removed circuit breakers has been verified as a safe design. Tables 4.6 illustrates the assume-guarantee pair generated for each design component.

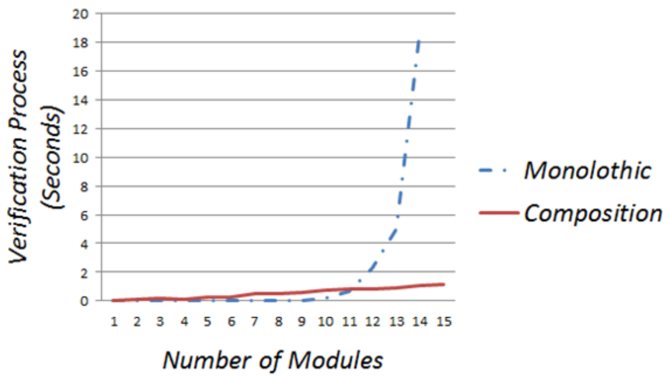


Figure 4.9: Verification Time of Monolithic Composition Vs. AGR Approach.

As illustrated, the generated assume-guarantee pair is critical in choosing alternative design solutions and migrating between different design architectures with relatively small effort. In addition, the generated assume-guarantee pair enables the

Table 4.6: Generated Assume-Guarantee Pair (Remove Unnecessary Circuit Breakers)

Improved EPS Design		
Component	Assumption	Guarantee
Battery	$[0, 1, 2]$	$[0, 1, 2]$
$ACResistor$	$[0, 1]$	$[0, 1]$
$Volt\ Meter_{240}$	$[0, 1, 2]$	$[0, 1, 2]$
$Current\ Sensor_{240}$	$[0, 1, 2]$	$[0, 1, 2]$
$Relay_{244}$	$[0, 1, 2]$	$[0, 1, 2]$
$Volt\ Meter_{242}$	$[0, 1, 2]$	$[0, 1, 2]$
$Invertor_2$	$[0, 1, 2]$	$[0, 1, 2]$
$Circuit\ Breaker_{266}$	$[0, 1, 2]$	$[0, 1]$
$Volt\ Meter_{265}$	$[0, 1, 2]$	$[0, 1, 2]$
$Relay_{272}$	$[0, 1]$	$[0, 1]$
$Current\ Sensor_{267}$	$[0, 1]$	$[0, 1]$
$Relay_{275}$	$[0, 1]$	$[0, 1]$
Fan	$[0, 1]$	$[0, 1]$
$Circuit\ Breaker_{280}$	$[0, 1, 2]$	$[0, 1]$
$Current\ Sensor_{281}$	$[0, 1]$	$[0, 1]$
$Relay_{284}$	$[0, 1]$	$[0, 1]$
$Volt\ Meter_{281}$	$[0, 1]$	$[0, 1]$
$DCResistor$	$[0, 1]$	$[0, 1]$

system designers to trace the requirements throughout the design architecture. The behavior of the system design is described in an assume-guarantee style specification: a component guarantees certain set of behaviors, given that its environment follows certain assumptions.

In addition, a performance case study is conducted by comparing the performance results of the proposed verification approach with a monolithical approach (Fig. 4.9) which represents linear growth for the proposed verification process, while an exponential growth is predicted for the monolithical verification process.

4.5 Conclusion and Future Work

In this paper represents system design requirements using a formal technique that allows for verification of these requirements from the early stages of system design by automatic execution of the specifications. This is especially important in proving the correctness of the system design, where it is critical to guarantee that the known interactions between system components do not violate any safety properties. Most often high level system design requirements are decomposed into component and (sub)system requirements which logically map to the architectural decomposition of the system. Therefore, proof of correctness through pre-verification of system components and compositional reasoning is made possible. The aim of compositional reasoning is to improve scalability of the design verification problem by decomposing the original verification task into subproblems. The simplification is based on the assume-guarantee reasoning that results in approximating the requirements which a component and (sub)system places on its operational environment to satisfy safety properties. The case study of the EPS design demonstrated the capability of the proposed verification methodology to perform virtual integration of system elements and proving system-level requirements from the constraints allocated on the components. As a result, a class of design flaws has been uncovered because of an integration failure that occurs when system components satisfy their requirements in isolation but not at the system-level.

The proposed approach models the behavior of composite components using LTS models of the primitive components and their safety properties, which are based on the structural model provided by the Modelica model of the system design. In addition, a fully automated compositional verification technique is used to determine the correctness of the design with regards to its requirement and generate pairs of assume-guarantee using a learning algorithm. Experimental results showed the effectiveness of the compositional reasoning approach in reducing the complexity of the verification process by using modularity and abstraction. In addition, we showed how a deductive verification tool such as LTSA combined with LTS models can be used for verification of finite-state hardware system designs. The compositional verification helps in breaking a large complex system design into smaller parts whose ŞverificationŤ can be checked in order to prove that the safety property of the components and the (sub)system holds. The assume-guarantee approach which is based on a learning algorithm [14], produces and refines assumptions depending on failure propagation paths and queries, the verification process is assured [14] to terminate. In addition, the algorithm returns counterexamples which include failure propagation information in the early stages of conceptual design. Another advantage of the proposed approach for verification of engineered systems is its independence from human intervention and expert user in devising the appropriate assume-guarantee pair. The experiment in this paper provided strong evidence in favor of this line of research.

The future work will focus on the design verification and analysis of complex engineered systems with software controlling the hardware. The design of such systems requires collaboration between experts from different design domains. Therefore, there is a pressing need for considering the complex interdependencies among components and (sub)systems during system design and formally defining their interacting behavior. These interacting behaviors are important since there are possible cases where these systems have conflicting requirements or objectives, therefore, understanding and verifying these underlying interactions is crucial.

In future work, it is intended to expand the approach discussed in this paper to examine the learning algorithm and its generated assumptions to determine the most reliable design architecture of the redundant systems. In addition, it is our goal to investigate different aspects of fault tolerant system design requirements while taking into account automatic injection of multiple failures and reasoning about different types of recovery strategies.

Chapter 5: Resiliency Analysis For Complex Engineered System Design

Hoda Mehrpouyan, Brandon Haley, Andy Dong, Irem Y. Tumer, Chris Hoyle

Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing

Accepted May 2014

5.1 Introduction

Conceptual design is the earliest stage in the overall process of engineering system design. Past research efforts [1, 2, 3, 4, 5], have recognized the significance of utilizing fault-tolerance analysis during the conceptual design phase. However, anticipating component failure rates and system performance is difficult as detailed knowledge of system components and their performance criteria are not yet available. Therefore, it is important to develop fault-tolerance engineering tools that can be used during the early design of complex systems because of the inherent uncertainty in the performance of individual components and their interaction effects during the product life cycle cost [6]. Robustness in this context means operation of the system within the designed performance variance under all ranges of environmental conditions experienced in the field. For the engineered system to be robust, the design of the system is required to be robust, meaning that the system is able to function under the full range of environmental conditions that may be experienced during system operation [7]. Resiliency is recognized as maintaining system functions despite the existence of failures [8]. This differs from traditional definitions of robustness because resiliency deals with the functional response of a system. As designers, it is important to be sure that these systems are able to perform the functions they were designed to perform; something that robustness does not strictly deal with. Instead, robustness correlates to the ability of a system to produce performance characteristics despite the presence of these internal and external stimuli. Complete functionality of a complex engineered system does not necessarily have to be maintained for the system to be considered robust.

The main purpose of system reliability analysis is to determine the weakness of a design and to quantify the impact of component failures. The resulting analysis provides a numerical rank to identify which components are more important to system reliability enhancement or more critical to system failure. Design reliability analysis methods introduced in the research literature, such as the Function-Failure Design Method (FFDM) [9], the Functional Failure Identification and Propagation (FFIP) [10], and decomposition-based design optimization [11, 12] have begun to adopt graph-based approaches to model the function of the component and the flow of energy, material, and signal (EMS) between them. This work extends this idea to demonstrate the effect of the design architecture on the robustness of the system being designed.

In the past several years, scientific interest has been devoted to modeling and characterization of complex systems that are defined as networks. Such systems consist of simple components whose interactions are very basic, but their large-scale effects are extremely complex, (e.g., protein webs, social communities, Internet). Numerous research studies have been devoted to the effect of network architecture on the system dynamics, behavior, and characteristics. Since, many complex engineered systems can be represented by their internal product architecture, their complexity is dependent on the heterogeneity and quantity of different components as well as the formation of connections between those components. Because of this, system properties can be studied by graph-theoretic approaches. Complex networks are modeled with graph-based approaches, which are effective in representing components and their underlying interactions within complex engineered systems.

Research findings by Ash et al. [13] suggest that modular systems are less robust even though their individual components are designed with high robustness. Modularity describes the topology of a system or network. Modularity in a system is rather straight-forward. A system is modular if components or subsystems can be isolated from the greater system without compromising the structure of the rest of the system. For instance, a modular system topology is one that allows a component or subsystem to be removed without first removing many others. Networks are a little more difficult. A network is said to be modular if there are high concentrations of high connectedness in the network separated by low connectedness between 'modules'. Bagrow et al. [14] confirm these finding and further explain that the high robustness of modular systems is only possible if the components' failure can be isolated to their modules. Furthermore, Hölttä et al. [15] prove that while the hierarchical modular structure improves the system's robustness, excessive use of modularity results in loss of performance. On the other hand, there are studies [6, 16] that support the increase of modularity in the design of complex engineered systems. Therefore, in order to design a robust system and to recommend or oppose the modular physical system architecture it is utterly important to understand the architectural properties of complex engineered systems and the effect of design architecture topology on the propagation of failures within a complex engineered system.

In this research, we adopt approaches from graph theory and social network analysis to understand the robustness of the design architecture of a complex engineered system. Specifically, this paper shows the relationship between graph spectral theory eigenvalue analysis and how optimizing (or altering) the graph of a system will change system connectedness, thereby changing the robustness of the system. To accomplish this objective, the network model of a safety-critical engineered system in the context of complex network theory is constructed. Its network properties are calculated to determine system robustness. Constructive design architecture change recommendations are made to optimize the system robustness.

5.2 Background

Eliminating the likelihood of failures, and should failures occur, ensuring the continued operation of the system within a safe performance envelop until repairs can be made, are of paramount importance in mission critical complex engineered systems. To avoid the failures of critical components, setting aside the problem of identifying critical components, the engineering design literature recommends techniques such as Failure Mode Effects Analysis (FMEA) [17] or a Function-Failure Design Method [9] among many. While these techniques have proven useful where knowledge of failure modes and effects can be predicted, their most significant weaknesses are that they can neither readily handle interaction effects of failures nor identify the most vulnerable components without significant prior knowledge. While methods such as the FFIP technique address this issue [10], significant expertise of engineers and a knowledge base of previous products are still required.

In contrast to techniques relying on prior knowledge, the network topology analysis and biological concepts of resilience hold promise for addressing this problem in the engineering design domain. The study of network topologies provides interesting insights into the way that complex engineered systems are designed. Numerous studies [18, 19] have attempted to measure the resilience of complex networks. In the design of networks, the design philosophy is not to predict that failures will occur, but, rather, to design with the knowledge that failures will occur - that is, that nodes will fail and external 'attacks' on the network may happen. The challenge for the network designer is to ensure that the network continues to operate, or fails gracefully, even under such circumstances. The results of these efforts conclude that many complex systems exhibit a surprising degree of tolerance against failure in a specific class of networks called scale-free networks [20]. A scale-free network is an inhomogeneous network in nature, meaning that a significant number of nodes have very few connections while a small number of particular nodes have many connections. The inhomogeneous feature of a scale-free network allows for higher failure tolerance under random failure of nodes, but the network is more vulnerable to failure when the most highly connected nodes fail [21, 22, 23, 24]. In the case of designing resilient complex engineered systems, the design architecture can be modeled as a complex network and their resilience optimized by ensuring that critical components (nodes) are less vulnerable to failure while preserving the interconnectedness of interdependent components.

There are two concepts that are most relevant: contagion spread and failure tolerance. A contagion spreads by altering the states of nodes, which is not dissimilar from the situation of a degraded flow from a component altering the performance of interdependent components. Usually, this is described as degradation in system performance, and robust systems are those that maintain performance within a tight tolerance despite perturbations. However, engineered system designs are naturally different from models such as the internet and World Wide Web representations of complex networks. As a result, the appropriate network representation is critical to the success of modeling engineered system design, since the representation affects the accuracy and efficiency of the calculation for system modularization and optimization. In order to evaluate different system design architectures for any given design problem, the graph theoretic formulation must not depend on any particular design architecture. Therefore, a general and precise analytical model such as a Non-Linear Dynamical System (NLDS) that uses a system of probability equations [?] for accurate modeling of viral propagation in complex networks can be used to investigate the behavior of failure propagation in complex engineered systems. This approach examines the propagation behavior via a number of stochastic contact trials per unit time, where the infection expands at a constant rate from an initially infected vertex. In addition, existing research literature on the analysis of disease epidemic spreading [25, 26, 27] and dynamics of information spreading in social networks [28, 29] are focused on modeling the failure propagation in complex engineered systems. Despite the fact that the two models share similar features, they are very different. For example, the disease-spreading model is based on the physical contact between individuals in a social network. Many factors such as biological characteristics of both the carrier and infectious agent play an important role in the mathematical model of the spread. However, information spreading is possible through non-physical contact and via the use of communication infrastructures, also the decision of whether information should be spread to more individuals or not is made by individuals.

Consequently, the paper focuses on the epidemic spreading of diseases, and these types of models inspire the proposed model.

5.3 Methodology

The initial part of the research focuses on producing synthetic networks that model real-world system design. In order to evaluate each design, a Modelica-based structural model [30] is created and converted into a graph representation of the system's design architecture. The network is modeled by a connected graph $G = (V,E)$ which is a collection of vertices V (also called nodes) with edges E between them. In this context, components of complex engineered systems are modeled as nodes of the graph and the connections between these components are the graph edges. These graph representations are then be used as a tool to convert each design into an adjacency matrix of nodes (components) and edge connections. Assuming $\mathbf{A}$ signifies the adjacency matrix of an engineered system under study with n components, A is defined as follows:

$$\mathbf{A}_{ij} = \begin{cases} 1 & \forall\ [(\text{i,j}) \mid (\text{i} \neq \text{j}) \text{ and } (\text{i,j}) \in \Lambda] \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

where Λ symbolizes the set of components. $\mathbf{A}$ is a square symmetric matrix with diagonal entries of zero. the edge connections between components can be defined topologically or by Energy, Material, and/or Signal (EMS) relations. A topologically defined graph has components which are connected with physical justifications. For example, if two components are physically connected together within a design, they are connected within the graph. EMS related connectivity rules describe the energy, material, and/or signal flows between components. Therefore, if two design components share a flow variable (or a flow relationship), they would be *"connected"* within the graph and represented with a **"1"** within an adjacency matrix.

In addition, a degree matrix called **D** is used to define the number of connections associated with a specific node, or component and is defined based on the following:

$$\mathbf{D}_{ij} = \begin{cases} d_i & \text{degree of component i when i = j} \\ 0 & \text{when i = j} \end{cases} \tag{5.2}$$

$\mathbf{D}_{ij}$ is an $n \times n$ diagonal matrix where d_i is the degree of the vertex i recognized as the total number of edges that touch the vertex. Hence, every engineered system can be represented elegantly by graphical models. Then, the Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

$$L_{ij} = \begin{cases} d_i & \text{when i = j} \\ -1 & \text{when i = j and i is adjacent to j} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

The Laplacian matrix is a square, symmetric, and positive semi-definite matrix. As a result, it has nonnegative eigenvalues which are ranked using an index in ascending order:

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \tag{5.4}$$

In reviewing the literature in algebraic graph theory [31, 32, 33], the second smallest eigenvalue of the Laplacian matrix has appeared as a critical parameter for robustness

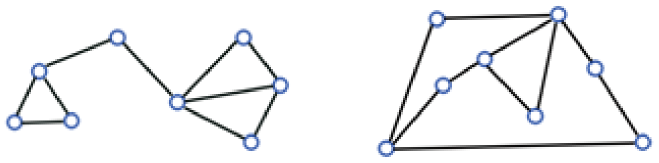


Figure 5.1: Even though both graphs have the same degree sequence, the graph on the left is considered weakly connected. On the left the algebraic connectivity equals 0.238 and on the right 0.925.

properties of dynamic systems that operate as networks. The second smallest eigenvalue of a Laplacian matrix is known as the algebraic connectivity. The algebraic connectivity describes the average difficulty to isolate an individual node (component) from the rest of the system (Fig. 5.1). Because the algebraic connectivity of a graph increases with increasing node and edge connectivity, a higher algebraic connectivity will result in an increased number of paths between nodes. This inherently means that networks with higher algebraic connectivity are more robust [32]. Additionally, another such parameter exists in the literature called spectral radius. In the review of literature from network theory, the spectral radius is the largest eigenvalue of the adjacency matrix [34]. Jamakovic et al. [34] conclude that a smaller spectral radius results in higher system resiliency against failure propagation throughout the system compared to other networks of similar average node degree. As this value is based on the evaluation of the eigen-spectrum of a single, unique characteristic equation, it is important to note that the value is not useful unless it is compared to another graph of similar size. It does not have a defined range of values. The spectral radius provides a high level analysis to compare two or more graphs.

The number of modules present in a network is determined using the eigenvalues of the adjacency matrix representation of the design [35, 36]. To determine the number of modules, the eigenvalues of the adjacency matrix are ordered in descending order. The differences between the ordered eigenvalues define the number of modules in the system. If k corresponds to an eigenvalue of the adjacency matrix, the maximum difference is between the k^{th} and $k^{th} + 1$ eigenvalue. The number of modules is the k value where that difference is the greatest. Given the relative quantity of modules within a design, this information can be used for insight into the robustness of a given design topology as well as the resilience to attack propagation for a specific design.

The following constraints are defined while modeling the system under design:

1. A component is not connected to itself, meaning that the diagonal of the adjacency matrix is a diagonal of zeros.

2. A system is represented as a connected system, therefore there is no isolated component (or set of components) with no connections to any other components.

5.3.1 Maximizing the Design Robustness

This section demonstrates the effect on the design architecture from maximizing the algebraic connectivity for system robustness. A random, generic system is represented as a network. This system is not meant to describe possible real-world system architectures, but rather to show how changes in network topology manifest in graph analysis metrics. A genetic algorithm is developed to iterate through a random network and change the connections between nodes (components) within an adjacency matrix. This is done by maximizing the algebraic connectivity of the network in each iteration. Each generation (iteration) represents a system architecture that experiences binary cross-over and mutation events to produce the next generation. The system under design is modeled in binary values with values of 1 representing a connection between two components, and 0 otherwise. The developed algorithm performs evaluations on a binary bit string of characters, which represent the connections between two components. Even though the Laplacian matrix is used to define the algebraic connectivity, the adjacency matrix is manipulated in order to iterate through the design space. The design space is defined by combinations of generic system components, where each component is capable of having a connection with any other component. This results in all possible system design candidate architectures including some infeasible architecture. The evaluation process is computed on a sequence of design variables, where every possible connection between one component and another is a design variable, represented in a binary string of 1 or 0 characters similar to the following:

$$Design = [101101010101011111000...] \tag{5.5}$$

For a more conventional problem, such as a design problem with defined design variables such as length, width, height, mass, etc., the algorithm manipulates the string so that the binary address of the design variables is changed from generation to generation. This is especially useful for a discrete problem that contains only a handful of possible design architectures. Fig. 5.2 depicts the steps of the algo-

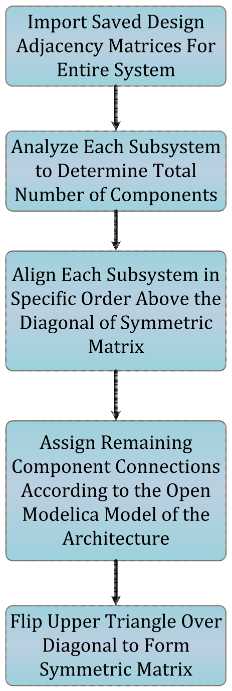

Import Saved Design Adjacency Matrices For Entire System
Analyze Each Subsystem to Determine Total Number of Components
Align Each Subsystem in Specific Order Above the Diagonal of Symmetric Matrix
Assign Remaining Component Connections According to the Open Modelica Model of the Architecture
Flip Upper Triangle Over Diagonal to Form Symmetric Matrix

Figure 5.2: Algorithm steps for computing the resiliency of the system under design.

rithmic process to compute the algebraic connectivity of the system under design. The maximization of the algebraic connectivity in the genetic algorithm produces adjacency matrices with higher component degrees, representing a higher average number of connections per component and therefore a higher algebraic connectivity, as expected.

5.3.2 Design Topology and Its Effect on Failure Propagation

The second part of the research determines how design architecture affects the propagation of failures throughout an engineered system. System robustness and resistance to topological failure propagation help to describe how a complex engineered system responds to internal and external stimuli.

The cascading failure is modeled as a Contact Process (CP), introduced by Harris [37], and has wide applications in engineering and science [38, 39]. A typical CP starts with a component in its failure mode, which affects the neighboring components at a rate that is proportional to the total number of faulty components. For such a system with n components, given any set of initially faulty components, the propagation of failure between components exists in a finite amount of time. This paper presents a reasoning method based on the length of time that the failure propagation is active in the system. With this information, system architectures can be identified which are resilient to the transmission of failures.

Spectral radius has thus far been presented as a quantity capable of producing insights into network resilience to propagating attacks. This metric is based off an evaluation of the network topology as a whole rather than the individual nodes providing avenues for attack propagation. In essence, the spectral radius metric does not capture a network's inherent ability to 'bottle-neck' failures with local topology. The following network propagation models allow for this type of analysis.

5.3.2.1 Non-Linear Dynamical System (NLDS) Modeling

The NLDS propagation model provides an indication for the length of time to full propagation according to the graph layout defined by an adjacency matrix. In the proposed model, a universal failure cascading rate β $(0 \leq \beta \leq 1)$ for each edge connected to a faulty component is defined. The model is based on discrete time-steps Δt, with $\Delta t \to 0$. During each time interval Δt, a faulty component i infects its neighboring components with probability β.

The proposed solution for solving a full Markov chain is exponential in size. In order to overcome this limitation, it is assumed that the states of the neighbors of any given component are independent of one another. Therefore, the *non-linear dynamical system* of 2^N variables is reduced to one with only N variables for the full Markov chain which can be replaced by Equation (5.8). This makes the large design problems solvable with closed-form solutions. Notice that the *independence assumption* is empirically very close to the full Markov chain [40] and does not place any constraints on the design network topology.

The probability that a component i is failed at time t is defined by $p_i(t)$ and the probability that a component i will not be affected by its neighbors in the next time-step is denoted by $\zeta_i(t)$. This holds if either of following happens:

1. each neighbor is in its nominal state.

2. each neighbor is in its failed state but does not transfer the failure with probability (1 - β).

With the consideration of small time-steps (Δt $\rightarrow$ 0), the possibility of multiple cascades within the same Δt is small and can be ignored.

$$\zeta_i(t) = \prod_{j:\ neighbor\ of\ i} (p_j(t-1)(1-\beta) + (1-p_j(t-1))) \tag{5.6}$$

$$= \prod_{j:\ neighbor\ of\ i} (1-\beta * p_j(t-1)) \tag{5.7}$$

In the above formula (5.6), it is assumed that $p_j(t-1)$ are independent from one another.

As illustrated in Fig. 5.3, each component at time-step t, is either Nominal $(\boldsymbol{N})$ or Failed $(\boldsymbol{F})$. A nominal component $\boldsymbol{i}$ is currently nominal, however can be affected (with probability $1 - \zeta_i(t)$) by one of its faulty neighbors. It is important to note that $\zeta_i(t)$ is dependent on the following:

1. The failure birth rate β.

2. The graph topology around component i.

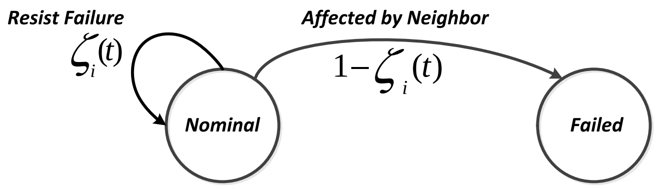


Figure 5.3: Transition Diagram of the Nominal-Failed (NF) Model.

The probability of a component i become faulty at time t is defined by $p_i(t)$:

$$1 - p_i(t) = (1 - p_i(t-1))\zeta_i(t) \quad i = 1...N \tag{5.8}$$

The above equation can be solved to estimate the time evolution of the number of faulty components (η_t), given the specific value of β and a graph topology of the conceptual design, as follows:

$$\eta_t = \sum_{i=1}^{N} p_i(t) \tag{5.9}$$

5.3.2.2 Epidemic Spreading Model (SFF)

In this approach, the theoretical model is based on the concept that each component in the complex system design can exist in a discrete set of states. The failure propagation changes the state of a component from nominal to failure or from failure to fixed. As a result, the model is classified as a susceptible - failed - fixed (SFF) model, in which components only exist in one of the three states. The design state fixed prevents the component from failing by the same cause. The densities of susceptible, failed, and fixed components, *S(t),* $\rho(t)$*,* and *F(t)* , respectively, change with time based on the normalization condition.

The proposed methodology is based on the universal rate (μ) in which the failed components are fixed in the design, whereas susceptible components are affected by the failure at a rate (λ) equal to the densities of failed and susceptible components. In addition, $\bar{k}$ is defined as a the number of contacts that each component has per unit time. It is important to note that the assumption made in this proposed model is based on the fact that the propagation of failure is proportional to the density of the faulty components. Therefore, the following differential equations can be defined:

$$\frac{dS}{dt} = -\lambda\bar{k}\rho\mathrm{S} \tag{5.10}$$

$$\frac{d\rho}{dt} = -\mu\rho + \lambda\bar{k}\rho\mathrm{S} \tag{5.11}$$

$$\frac{dF}{dt} = \mu\rho \tag{5.12}$$

In order to estimate *S(t)*, the initial conditions of *F(0)* $=$ *0* (no design fix is implemented yet), *S(0)* $\simeq$ *1* (almost all the components are in their nominal or susceptible modes) , and $\rho(0) \simeq 0$ (small number of faulty components exist in the initial design) is assumed. Therefore the following can be obtained for *S(t)*:

$$S(t) = e^{-\lambda\bar{k}\rho\mathrm{F(t)}} \tag{5.13}$$

In order to address the contact process in an engineered system, a general connectivity distribution *P(k)* is defined for each design network. At each time step, each nominal or susceptible component is affected with probability λ, in the case of being connected to one or more faulty components. At the same time, every faulty component is repaired in the system design so they are resilient against a similar failure. It is assumed that the designers of the system fix the faulty components with probability μ. Because every component in an engineered system has different degrees of connectivity *(k)*, the time evolution of $\rho_k(t),\ S_k(t),\ and\ F_k(t)$ which are the density of faulty, susceptible, and fixed components with connectivity k at time t is considered and analyzed. Therefore the Equation in (5.13) can be replaced by the following:

$$S_k(t) + \rho_k(t) + F_k(t) = 1 \tag{5.14}$$

As a result, the global variables such as $\rho(t),\ S(t),\ and\ F(t)$ are expressed by an average over the different connectivity classes; i.e., $F(t) = \sum_k P(k)F_k(t)$.

The above equations combined with initial conditions of the system design at $t = 0$ can be defined and evaluated for any complex engineered system.

5.4 Case Study

The paper explores the design space for two case studies to demonstrate the features of graph spectral theory on complex engineered system design. The Advanced Diagnostics and Prognostics Testbed (ADAPT) is designed based on the requirement to generate, store, distribute, and monitor electrical power in an exploration vehicle. The Electrical Power System (EPS) testbed developed at NASA Ames Research Center is used as an example to describe the spectral analysis process while the Ramp System of an Infantry Fighting Vehicle (IFV) is used to provide comparisons between designs. Additionally, two failure propagation models are implemented on the Infantry Fighting Vehicle (IFV) network models in an effort to examine their topological structure for resilience to failure propagation. These models, the Non-Linear Dynamical System Model and the Epidemic Spreading model, are presented with two cases of different failure origins, a highly connected component (an electrical ground node) and a minimally connected component (an electrical circuit breaker node).

5.4.1 Graph Spectral Theory

Spectral graph approaches were utilized on the ADAPT testbed and the IFV ramp networks. This analysis includes an evaluation of network robustness from algebraic connectivity, overall network resilience to propagations from spectral radius, and an evaluation of modularity.

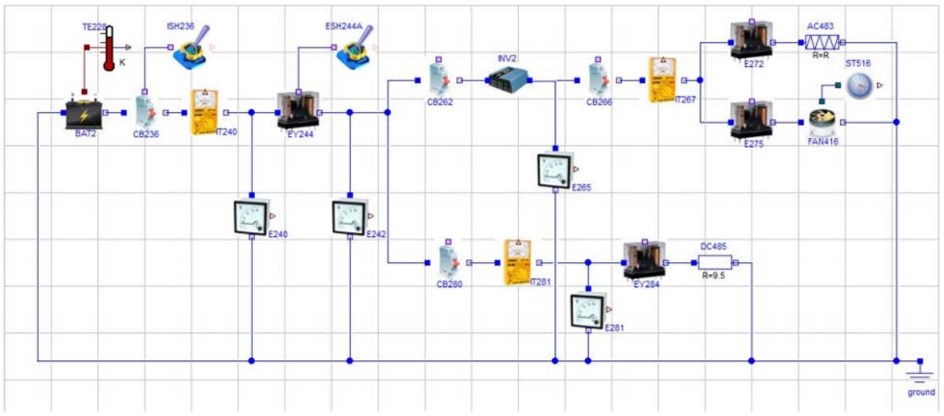


Figure 5.4: Model of the existing electrical power system design architecture.

5.4.1.1 ADAPT Electrical Power System (EPS)

Fig. 5.4 displays the Modelica [30] representation of an existing design of an EPS [41]. Modelica is a language for hierarchical object oriented modeling of engineered systems, which was developed through an international effort. The EPS model contains a power source connected through a series of relays to an inverter and several loads consisting of a large fan, a direct-current (DC) resistor and an alternating current (AC) resistor. A series of four AC or DC voltage sensors and three current transmitters measure the voltage and current at different points throughout the circuit. The Modelica representation of the system is converted to a network representation as described in Section 3. The generated network is used to convert the system into an adjacency matrix of nodes (components) and edge connections. Including the electrical ground, the EPS system consists of twenty-five nodes or "components". These nodes and edges define the connectedness of the system, or the design architecture of the system.

The first step is to create the adjacency matrix of the network representation of the EPS system. The first row depicts the battery and its connections to other components in the system. The first column of the first row is represented by a set of zeros, since the battery is not connected to itself. The second column of the first row is assigned one representing the battery's connection to the circuit breaker. Therefore, each row in the matrix represents a component in the design and each column signifies the component's connections with other components in the system. Then the degree matrix and Laplacian matrix for the EPS, which resulted from Equations (2) and (3) are created. A table of the eigenvalues resulting from the adjacency matrix and the Laplacian matrix can be found in Table 5.1.

Table 5.1: Eigenvalues generated in EPS Design Architecture

Adjacency Matrix Eigenvalues	Laplacian Matrix Eigenvalues
-3.236	1.457E-15
-2.196	***Algebraic Connectivity*** $\longrightarrow$ ***0.247***
-1.978	0.338
-1.902	0.479
-1.574	0.535
-1.319	0.659
-1.222	0.669
-1.074	0.899
-0.808	1.152
-0.727	1.219
-0.455	1.540
-0.204	2.010
0.000	2.264
0.349	2.412
0.397	2.640
0.745	2.920
0.802	2.930
1.084	3.533
1.281	3.908
1.353	3.947
1.538	4.066
1.824	4.505
1.931	4.596
2.068	5.211
Spectral Radius $\longrightarrow$ ***3.324***	9.324

From the eigenvalues of the adjacency matrix, a spectral radius of 3.324 is computed. As stated previously, this number is not directly usable without a comparison to other designs. However, as this example is meant to show the process of converting a complex engineered system model to a network, the implications of the spectral radius will be discussed further with the comparison of the ramp models.

Table 5.2: Spec EPS Design Architecture

EPS System Results							
EPS Design ID	Nodes	Min Node Deg.	Max Node Deg.	Avg Node Deg.	Spectral Radius	Alg. Conn.	Modules
1	25	1	8	2.4800	3.3243	0.2473	11

As seen from Table 5.4, nearly half of the components within the EPS system are also modules. Since many electrical components can only be connected in certain configurations, the possible system design space is limited. The analyzed EPS has components connected in both series and parallel connections, with the parallel connections representing electrical modules. Alternatively, the algebraic connectivity of the system is rather high when compared to the Ramp designs, as discussed in the next section. This is mainly due to the properties of an electrical circuit. The EPS is modeled with a ground as a component within the system graph. An electrical ground must exist and be connected to the proper components in order to complete the circuit. As a result, circuits tend to be interconnected, which increases the average node degree of the graph and the algebraic connectivity. However, this relationship does not always apply. As will be seen in a Ramp design case study, a high average node degree does not always correspond directly to an increased algebraic connectivity. Some systems contain subsystems, which are independent of the overall system, but highly interconnected within their own subsystems (high average node degree). Therefore, if the rest of the system is sparsely connected, the isolated area of high interconnectedness drives the average node degree of the system up, while the algebraic connectivity remains low as it relates to the Laplacian of the overall system.

5.4.1.2 Ramp System of the Infantry Fighting Vehicle (IFV)

To demonstrate the benefits and scalability of using spectral graph theory on complex engineered systems, a ramp system of the Infantry Fighting Vehicle (IFV) is modeled and analyzed next. The modeled ramp is located at the rear of the IFV and used for the speedy exit and entry of the troops and the power-operated ramp is also fitted with a door. A hierarchical Modelica ramp model consists of an EPS, a mechanical ramp subsystem, a controller subsystem, and a crew subsystem.

Graph spectral analysis was conducted on three different design architectures of the ramp system. Fig. 5.5, and Fig. 5.6 illustrate the design options. Each design implemented in Modelica is a system of subsystems. Therefore, every "component" seen in the designs is actually a system of components making up a larger nodal percentage of the system as a whole. Each design consists of at least a unique EPS and mechanical subsystem. In addition, each design has modeled troops entering, exiting, and residing within the vehicle.

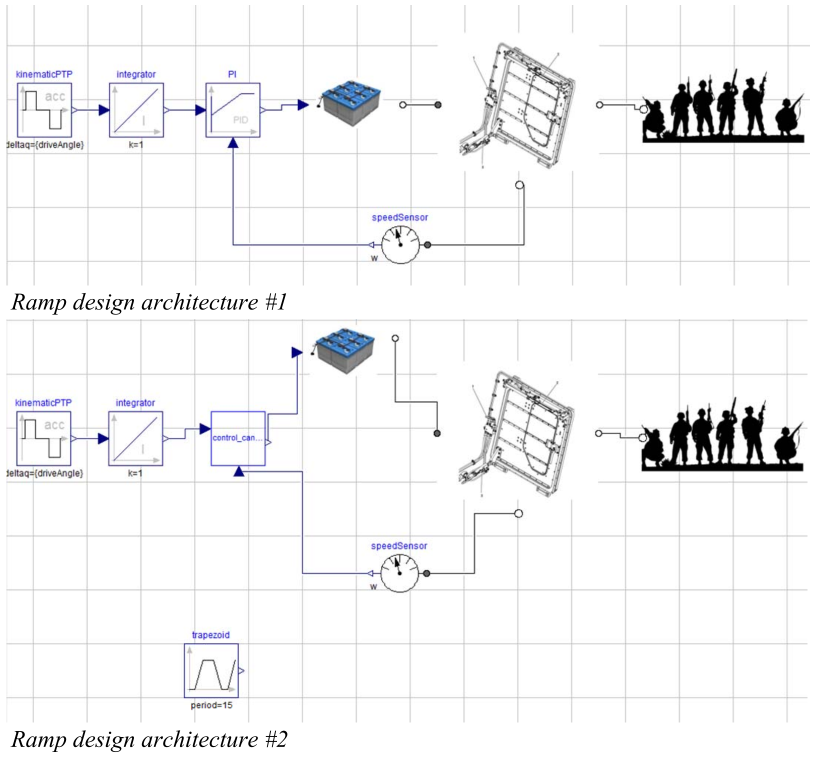


Figure 5.5: First and Second Ramp System Design Architectures

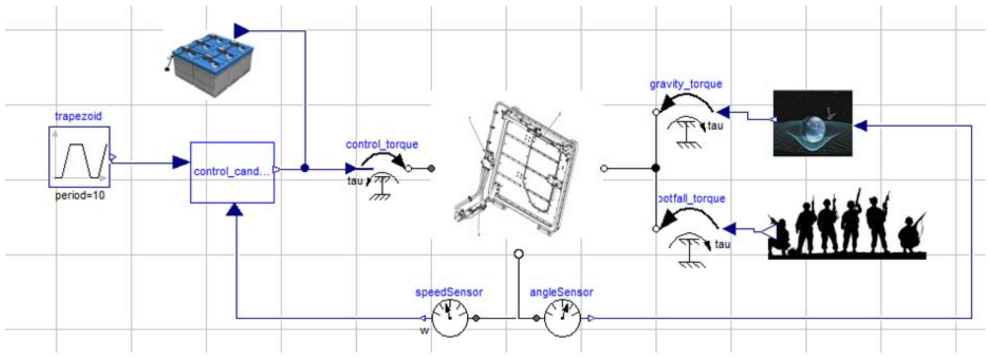


Figure 5.6: Third Ramp System Design Architecture with Design.

The graph representations of the three ramp designs are shown displayed in Fig. 5.7. In addition, Table 5.3 provides pertinent information concerning the three designs for use with both propagation models. Included is the design identification number to be used during the analysis, the number of nodes (components) contained within each design, the minimum degree, the maximum degree, the average degree, and the number of modules.

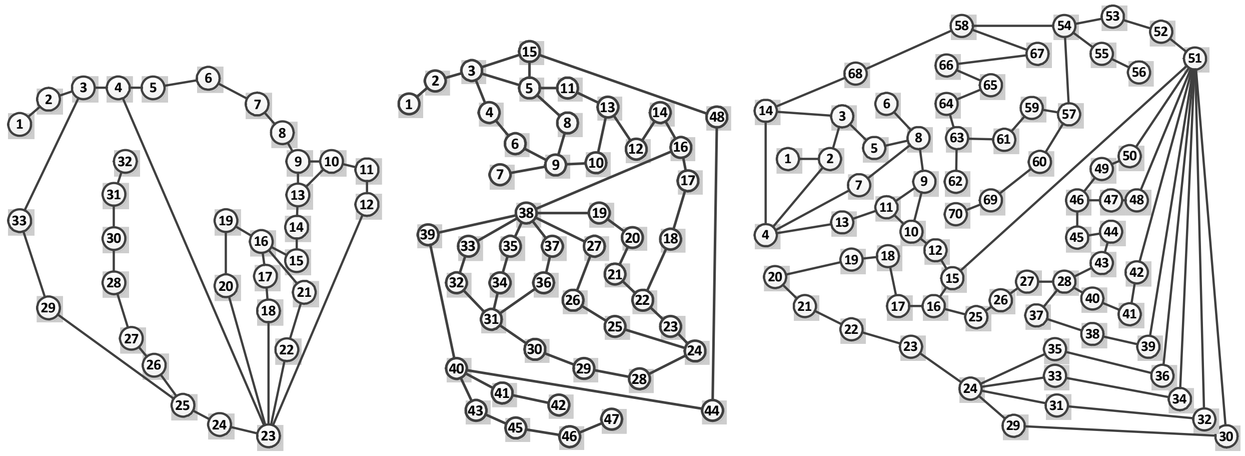


Figure 5.7: Graph Representation of the Ramp Design 1, 2, and 3

Table 5.3: Spec Ramp Design Architectures

Ramp System Design Properties							
Ramp Design ID	Nodes	Min Node Deg.	Max Node Deg.	Avg Node Deg.	Spectral Radius	Alg. Conn.	Modules
1	33	1	6	2.3030	2.8106	0.0479	10
2	48	1	7	2.3750	2.9474	0.0481	34
3	70	1	10	2.3714	3.3899	0.0295	47

Each ramp design is analyzed for failure propagation by evaluating the design architecture for the length of time to full propagation (NLDS) and for the breadth of propagation (SFF) when a failure is introduced. The third ramp design consists of the highest number of modules, yet has the lowest algebraic connectivity. This is an important insight, as it is commonly known that modularity in complex engineered systems is useful for system construction and maintainability, but the isolation of failures into a single module typically makes the system less robust, as shown.

5.4.2 Failure Propagation Models

Two failure propagation models are used to explore the design space of the ramp system of an infantry-fighting vehicle (IFV) to demonstrate resilience against cascading failure. Three complete ramp system designs will be assessed with a non-linear dynamical system propagation model (NLDS) and an epidemic spreading model.

Table 5.4: Initial Faulty Components in the Ramp Designs

The Node Number of the Faulty components in Design Graphs			
Faulty Components	Ramp Design #1	Ramp Design #2	Ramp Design #3
Circuit Breaker	5	17	16
Ground	23	38	51

Table 5.4 provides the information for failure origin utilized in this simulation. Node numbers have been provided which correlate to the node numbers used in Fig. 5.7.

5.4.2.1 Non-Linear Dynamical System Model (NLDS)

In order to gauge the degree of failure propagation in the design architectures (Fig. 5.7), an initial set of components in a state of failure is defined so the failure can propagate along the underlying graph structure of the architecture. For the sake of comparison, each topology has been compared twice, once with an initially failed, minimally connected component and once with a highly connected component. Additionally, in order to compare different architectures, each cascade is set to originate from the internal EPS subsystem, or electrical power system of each conceptual design. Specifically, a minimally connected circuit breaker and a highly connected ground terminal are selected as the failure origins. As it can be seen in Fig. 5.8 the population of the infected components with respect to time is the same for all three different design ramps. In the EPS sub-system of each ramp design, the circuit breaker has two connected components that can be infected. Therefore, all three designs propagate similarly until a component is failed which can cause a drastic increase in infected population size. This occurred near the sixth time step for each design. For instance, when the failure reaches the ground node of the EPS, the failure is able to spread much more quickly because the ground node is the most highly connected component in each design. The result confirms the expectation that a more highly connected component propagates failure to neighboring component more quickly, while a minimally connected component, such as a circuit breaker, results in slower failure propagation. As can be directly observed from Fig. 5.9, a failure originating from a more highly connected component (in this case the EPS ground) propagates much more rapidly. The NLDS model proves that more highly connected components spread a failure much faster. Therefore, nodal hubs, or very modular areas of a design are more detrimental to the rapid spread of a failure. However, simply having modular design structures does not suggest an inadequacy in resilience towards failure cascades. As can be seen from Table 3, a large percentage of the components

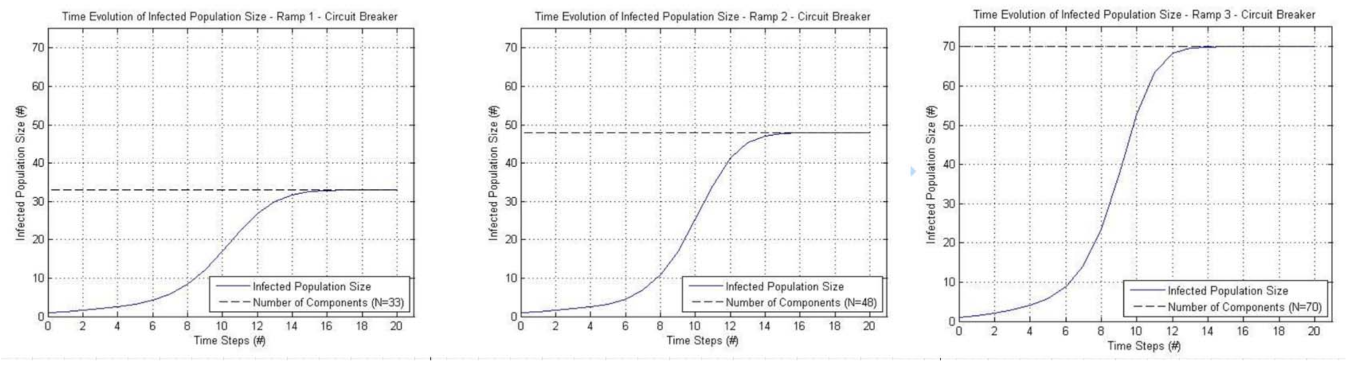


Figure 5.8: Time Evolution of Faulty Components' Population Size (Origin of Failure: Circuit Breaker).

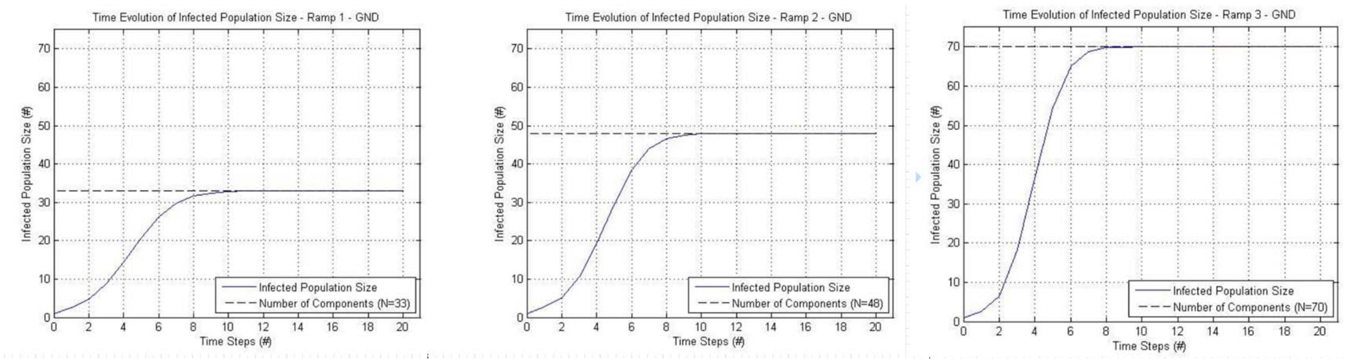


Figure 5.9: Time Evolution of Faulty Components' Population Size (Origin of Failure: Ground).

within each design are considered modular hubs. An important design aspect to note, however, is that there was no significant evidence for either initial failure state to suggest that one ramp design was more resilient to failure cascades than any others. This is based on the assumption that cascading time defines resilience and not infected population size. When the same, minimally connected circuit breaker has failed in each design, the time to full propagation for each design is approximately fourteen time steps. When the ground terminal is initially failed, the time to full propagation is approximately eight time steps. Additionally, the shape of each relationship, representing the progression of failed population size, does not indicate any design being more resilient to failure propagations.

5.4.2.2 Epidemic Spreading Model (SFF)

Unlike the NLDS model, the SFF epidemic spreading model is based on the idea that failure propagation can be stopped by fixing the faulty components. The SFF model operates by the spread of a failure from an initially failed component just as the NLDS model does. However, the SFF model is not a probabilistic model that is solely dependent on the architecture of an adjacency matrix as the NLDS model is. Instead, the SFF model requires a time step dependent simulation of the spread of a component failure. As with the NLDS model, a time step is regarded as sufficiently close to zero so that only the current population of failed components transmit a failure.

Each "faulty" component has an opportunity to infect a neighboring susceptible component in the next time step. In one time step, a component infects its connected neighbors according to a uniform failure probability. The simulation run for the SFF model was conducted at $\lambda = 10\%$. After a component has had an opportunity to infect its neighbors, the infected component would then be fixed in the conceptual design to resist the same failure according to the probability of failure removal $\mu = 10\%$. A repaired component is either considered faulty without the ability to transfer the failure to the neighboring components or is susceptible but resistant to the failures of its connected neighbors. Therefore, the cascading failure could be

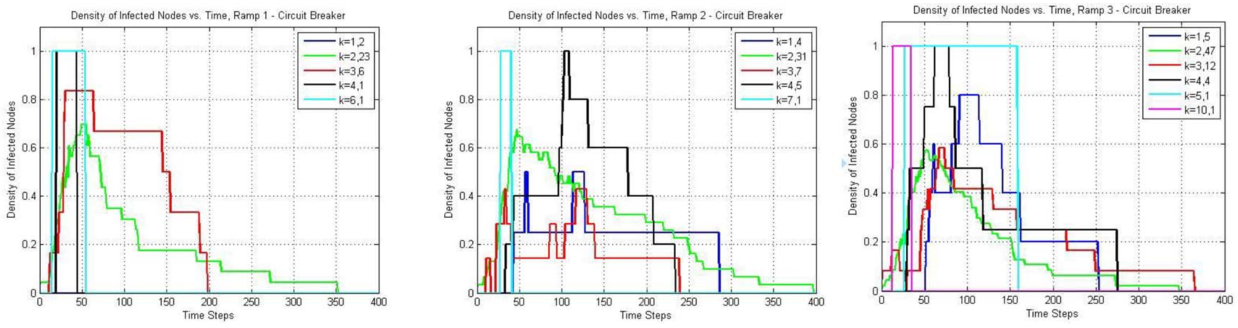


Figure 5.10: Time Evolution of Faulty Components Density for Three Conceptual Ramp Design (Origin of Failure: Circuit Breaker). (The k value is the indicative of the degree of the components followed by the number of components within that data set.)

stopped with the provision that enough faulty components become repaired in the design before they are able to fully propagate the failure. That is, propagations can be halted if all transmission routes are blocked by repaired components. The same designs were used with the epidemic spreading model as were used with the NLDS model. Additionally, the same initial failure conditions were used. An EPS circuit breaker was initially failed as a minimally connected component. A ground node was then initially used to propagate the failure as a highly connected component. Fig. 8 shows the epidemic spreading graphs for an initially failed circuit breaker within the EPS for each ramp design.

The same designs were used with the epidemic spreading model as were used with the NLDS model. Additionally, the same initial failure conditions were used. An EPS circuit breaker was initially failed as a minimally connected component. A ground node was then initially used to propagate the failure as a highly connected component. Fig. 5.10 shows the epidemic spreading graphs for an initially failed circuit breaker within the EPS for each ramp design.

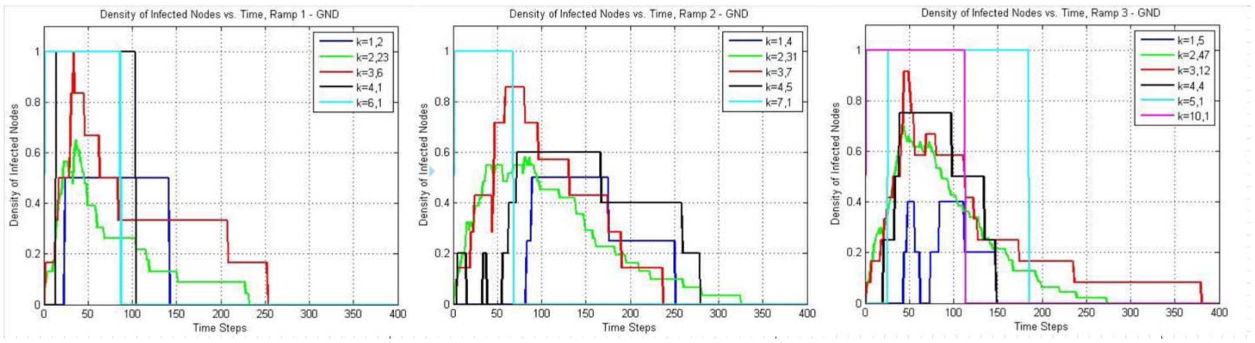


Figure 5.11: Time Evolution of Faulty Components Density for Three Conceptual Ramp Design (Origin of Failure: Ground).

In order to compare the time evolution of faulty component density in the three different conceptual ramp designs, a component with an equal number of connections from the EPS subsystems of the ramps is chosen as an initial faulty component. The reason for this is the fact that SFF failure propagation is based on connections, therefore the results must be reported in terms of faulty component density. As it is depicted in Fig. 5.10, each data set is representative of a set of components with the same degree, e.g., ramp 1: red colored data set represents six components in the system design with only three connections. Therefore, each set of components has a failure density ranging from 0 to 1; 0 means that no components of that degree are infected and a 1 means that every component of that degree being infected. Fixed components are not considered faulty. Consequently, a plot of faulty component density fluctuates intermittently between 0 and 1, but eventually settles at 0 as all failed components are fixed in the system design. In the legend of each graph, a k value is given which is indicative of the degree of the components followed by the number of components within that data set. When a minimally connected component, such as a circuit breaker, is chosen as a failure origin, it is compared to an initially infected highly connected component, such as a ground, and the failure spreads more slowly, as expected.

Fig. 5.11 illustrates the simulation results for an initially failed, highly connected ground terminal. The plots present more immediate increases in infection density, regardless of component degree when a highly connected component is failed initially. However, once the initial infection has passed, and the failure density begins to subside, the reduction of infection density is not dependent on architecture. This is because a stopped failure is repaired according to a uniform probability. This failure is not then passed between connected components.

5.5 Discussion

The first ramp design consisted of the fewest number of components. Each ramp design was highly modular, especially the third design, which is the most modular EPS. The third ramp design is a good example of why looking at the average node degree and algebraic connectivity independently is an unreliable exercise. As can be seen from the results in Table 2, the first two ramp designs are very similar. This is not the case with the third ramp design, which has a smaller algebraic connectivity with more modules. This is due to the third EPS design used with that ramp variant. When compared to the other two EPS designs, the third design has an algebraic connectivity half that of the other two. In addition, the third EPS design has more components than the others with a similar average node degree. However, the maximum node degree of the third design was greater than the other EPS variants.

Upon further examination, the third EPS design is found to have many components that are only connected to two components, one which supplies electrical current and one that receives current, making it simpler to make a component independent from the rest of the system. Additionally, a limited number of components are connected to many others. This small number of components is what drives the average node degree of the system up; making each EPS design appear similar by average node degree. However, because the algebraic connectivity is a measure of the difficulty in making a component independent of the rest of the system, the algebraic connectivity of the third ramp design remains low because of the number of components that have a fewer number of connections. The low algebraic connectivity is consistent with the recommendation for modular physical system architectures that may have the unintended downside of making the systems less tolerant to failure.

After both models were applied to each of the three ramp designs, the NLDS model did not show any relevant evidence to suggest that any one ramp design (graph layout) was more resilient to propagations than the others. However, conclusions can still be drawn from the results and are discussed below. The length of time to full propagation was not significantly different between designs. The NLDS model can adequately identify those design components that are critical to a system and whose failure would cause shutdown of the whole system, as can be seen by the differences in failure origins. Conversely, the SFF model can be used to compare different conceptual design architectures for resilience to propagation. This can be done by analyzing how a failure propagates through a system and then fixing failed components to inhibit the propagation of the failure. The SFF model determined that ramp design 3 was more resilient to specific nodal attacks, as both simulations indicated a similar infection breadth. This result is consistent with the observation of spectral analysis, since the design 3 is more modular compared to the other designs. Therefore, it has a high robustness only because the failure of components can be isolated to its module.

A couple of telling conclusions can be drawn from the result of the case study. Firstly, the NLDS models showed that connectivity plays a major role in how fast an epidemic spreads. A few components with a higher degree increase the speed of infection throughout a system. This was conclusively shown because the third ramp design, having a few nodes with a higher degree than the other designs, will propagate a failure faster. Secondly, larger systems will lessen the impact of random or targeted attacks. The ramp system designs showed this because the normalized percentage of failure began to equalize between components with fewer connections and components with more connections that were used as failure origins when the system size increased. Both conclusions provide insight into design architectures that can be more resilient to failures.

5.6 Conclusion and Future Work

Establishing robustness during the conceptual design phase is a difficult yet important aspect to the design of engineered systems. Utilizing complex network theory in conjunction with spectral analysis has provided useful insight into the design of robust complex engineering systems. Spectral analysis provides valuable metrics in quantifying certain aspects of complex networks. These metrics are algebraic connectivity, modularity, and spectral radius. As stated in this paper, the algebraic connectivity represents the difficulty in making one node independent of the rest of the system. A higher algebraic connectivity denotes increase in components' connectivity and higher robustness of the overall system. Because of the close correlation between complex networks and complex engineered systems, algebraic connectivity is a good metric to be considered to determine the resilience of the architecture of complex systems. To determine the resiliency characteristics of complex engineered systems, two case studies involving complex engineered systems were analyzed using spectral analysis. Utilizing the algebraic connectivity as the main analysis metric, both case studies provided evidence for the validity of using graph spectral theory on complex engineered systems.

Further, in the second part of the research based on the two propagation models, a Non-Linear Dynamical System (NLDS) model and an epidemic spreading model are developed for use during the early design of complex systems. From the two models, equations are provided to model the propagation characteristics of failures in complex engineered systems. The NLDS propagation model provides an indication for the length of time to full propagation according to a graph layout. The SFF epidemic spreading model provides an indication of the extent of a cascade according to a graph layout. While both models provide an indication into properties relating to failure propagation, they both require the accurate modeling of a complex engineered system as a graph. This is no trivial task. Because the graph of a system is dependent on how connections are defined, it becomes increasingly important to develop a standard methodology for modeling. In this paper, the ramp designs were modeled based on physical connectivity. To more accurately analyze these propagations, additional research would analyze the effect on a complex engineered system if its graph were created using different, and perhaps more complicated metrics of design dependency. Such justifications could include expanded physical connections that are inclusive of secondary component interactions. They do not necessarily have a physical connection interface and may be produced as a consequence of system operation. This could include heat, noise, and vibration related interactions, among others. In addition, justifications related to energy, material, and signal (EMS) flows would be a necessary next addition to analyze these models for applicability with complex engineered systems. EMS flow relations would create connections between components that share a flow. For instance, two components would be connected as a part of a thermodynamic process if the same working fluid travels from one component to the other.

Analyzing multiple adjacency matrices created with various connection justifications per design would provide a more complete look at a design's resilience to propagations. This would add computational expense in the analysis of the design and designer effort to create the matrices. An automated method to create the adjacency matrices would solve this issue; however, this would require an interface between a design tool such as a CAD suite and a matrix creator. In order to validate the use of these models, performance data of an engineered system that is operating in a state of failure is a necessary next step. By analyzing the time dependent response of a system under failure, the propagation properties predicted by these models can be verified.

Chapter 6: Towards A Framework For Resilient Design Of Complex Engineering Systems

Hoda Mehrpouyan, Irem Y. Tumer, Chris Hoyle, Andy Dong, Dimitra Giannakopoulou, Guillaume Brat

Journal of Research in Engineering Design

Submitted Jun 2014

6.1 Introduction

In recent years, technological advancements and a growing demand for highly reliable complex engineered systems, e.g., space systems, aircrafts, and nuclear power plants have made the safety assessment of these systems ever more important. Moreover, concurrently, the complexity of such systems makes it more challenging to achieve a design solutions that satisfy safety requirements of such complex systems. Traditional safety analysis techniques such as Failure Model and Effect Analysis (FMEA) [1], and Fault Tree Analysis (FTA) [2] are accepted methods that have been applied to predict the safety of system design. However, these methods require detailed knowledge of the system under design and can only be performed at the later stages of the design process. This late analysis results in missed opportunities for integrating the safety requirements at the early stages, while also causing extra effort and cost to modify the design at the later stages. Other limitations of FMEA and FTA include being significantly time consuming, expensive, and error-prone since the safety analysis are performed manually.

In order to predict the safety of such systems, within both the design theory [3, 4, 5] and systems engineering [6, 7] communities, model-based approaches have been proposed to address some of the shortcomings and limitations of the traditional methods, e.g., moving from textual definition to computable, reusable, and verifiable models. Among the advantages of the Model-Based Systems Engineering (MBSE) approach according to Bozzano et al. [8] is the ability to automate some parts of requirement verification and safety analysis. This is achieved through automatic generation of failure propagation paths, generating consistent results, and most importantly, providing the ability to integrate design and safety process around a central system model.

Based on these advantages, this paper presents a model-based safety specification and verification approach applying compositional reasoning and model checking algorithms. In addition, this paper describes an approach commonly used with complex networks to study the failure propagation in an engineered system design. The goal of the research is to synthesize and illustrate system design characteristics that results from possible impact of the underlying design methodology based on cascading failures. Further, identifying the most vulnerable component in the design or system design architectures that are resilient to such dissemination of failures provide additional property improvement for resilient design. Furthermore, it addresses the issue of formally specifying and formulating the design architecture that is resilient to component failures by exploiting redundancy. The application of component redundancy improves system reliability but also adds cost, weight, size, and power consumption. Therefore, it is vital to minimize the number of redundancies. The safety analysis and verification process proposed in this research examines the number of choices to determine a best way to incorporate redundancy into the design. It is *assumed* that the impact of failure on the system is known and it is *guaranteed* that the system is able to correctly satisfy its design requirements in spite of the occurrence of such a fault with the proper use of component redundancy.

The remainder of this paper is structured as follows: section *2* presents the background and related research on failure analysis techniques in the early stages of system design, while discussing their strengths and weaknesses. In addition, the definition of *compositional reasoning* and its commonly used terminologies and operators are addressed in section *2*. In section *3* an overview of the step-by-step implementation of the compositional reasoning algorithm on the components of the design architectures is explained. Section *3* describes the proposed modeling methodologies: 1- Non-linear dynamical system model, and 2- Epidemic spreading analysis models. Further, section *4* outlines the application of the proposed methodology in the analysis and verification of the safety properties of the quad-redundant Electro Mechanical Actuator (EMA) system design. The paper ends with conclusions and future work.

6.2 Background

The first step in specifying and formulating a complex systems design requires modeling and reasoning about its behavioral and functional characteristics. A commonly used formalism [7, 9, 10, 11, 12] to model and reason about complex systems design promotes the notion of functional decomposition. Decomposition represents efforts to break the design problem into smaller design elements and identify possible solutions to the design problem. Therefore, the required functionality of the system can be modeled before an actual design solution is implemented. Once the functional requirements of the design are identified, each function can be represented as an input flow, output flow, and internal behavioral states. Further, in the work of Bhatta et al. [13], the internal behavior of each function is modeled as a series of state transitions. However, Cellier et al. [14] recognize that these types of explicit modeling of input to output transactional representation are not appropriate for modeling the behavior of a complex engineered system design. Instead, several modeling platforms such as Modelica [15], Simulink [16], and bond graph related modeling [17] have been proposed to abstract the behavior of the component of the design independent from the rest of the system. Then, these models of components are automatically assembled to characterize the complete system. These types of approaches are known as component-based or object-oriented modeling which are based on the decomposition of the design into its constitutive elements.

While there are promising modeling techniques for abstraction and decomposition of complex systems' characteristics, there are few methods to prove the correctness of the design with regards to its safety requirements. One of the first approaches to formally map failures to functional losses and model the effect of failure on system design was the Function Failure Design Method (FFDM) [18, 19]. Since the FFDM is dependent on the historical failure propagation data, the approach is limited to analysis of the impact of a single fault. Therefore, the Risk in Early Design (RED) [20, 21] and Failure Propagation Analysis [22] methods were developed to overcome the limitation of FFDM approach. Finally, the Function-Failure Identification and Propagation (FFIP) [17] framework was developed to identify functional failure with the use of model-based reasoning methods. FFIP has proven to be effective in failure analysis and reducing the risk of failures; however, the FFIP framework is not proposed as a verification method, where the safety requirements are defined and analyzed to prove that a design satisfies them.

In addition, conventional techniques primarily deal with formulations of functional structures and conceptual design generation in response to system functional requirements. In most cases, these methodologies do not take into consideration the details of physical components, despite the fact that reliability research and industrial practice consider the detailed architecture of the physical components as a required input in the reliability analysis procedure [23].

In the past several years, scientific interest has been devoted to modeling and characterization of complex systems that are defined as networks [24, 25]. Such systems consist of simple components whose interactions are very basic, but their large-scale effects are extremely complex, (e.g., protein webs, social communities, Internet). Numerous research studies have been devoted to the effect of network architecture on the system dynamics, behavior, and characteristic. Since, many complex engineered systems can be represented by their internal product architecture, their complexity is dependent on the heterogeneity and quantity of different components as well as the formation of connections between those components. Because of this, system properties can be studied by graph-theoretic approaches. Complex networks are modeled with graph-based approaches which are effective in representing components and their underlying interactions within complex engineered systems. The vertices, representing the components of the product, are connected by a set of edges to denote the direct interaction between any pair of components [26, 27].

Within the framework of complex networks, studies have mainly concentrated on the spread of viruses in computer science [28, 29] and the spread of infections in complex population networks [30]. Of particular importance is the research devoted to the conditions under which failures spread quickly. Further, Boschetti et al. [31]

suggest that a graph theoretic formulation of complexity can be adopted to capture the structural design information in the system. Specifically, this is applicable to the connectedness of engineered systems. In order to evaluate different system design architectures for any given design problem, the graph theoretic formulation must not depend on any particular structure in the system design topology. Most introduced models in complex network literatures are designed to fit to a particular topological graph structure such as power-law, BA, and homogeneous; however, the two proposed models in this research have the benefit of being independent of any structural topology. Therefore, a general and precise analytical model such as a *Non-Linear Dynamical System* (NLDS) model that uses a system of probability equations [29] for accurate modeling of viral propagation in complex networks can be used to investigate the behavior of cascading failure in complex engineered systems. This approach examines the propagation behavior via a number of stochastic contact trials per unit time, where the infection expands at a constant rate from an initially infected vertex.

While the above approaches mostly concentrate on improving the design reliability through fault avoidance, application of component redundancy is another popular technique to improve reliability of the system design. Based on the literature review presented by Aspinwall [32], the vast majority of research efforts [33, 34] on reliability optimization, especially those focusing on a determination of optimal component redundancies require detailed knowledge of components, e.g., components' failure rates. Other redundancy optimization methodologies are based on weight coefficients [35] to allocate redundancy on the components of the system. Minimization of design cost subject to safety constraints [36] or maximization of design reliability under cost constraint [37] are examples of different sets of approaches. However, at the early stages of the design, selection of specific components has not been completed, therefore, components' cost, weight, or failure rate is unknown at this point. The only information available to the system designers at the conceptual level includes functions, behaviors, and a basic design structure.

In this paper, a model-based safety approach is proposed based on the *behavioral* models of design components, where behavioral specifications are associated with each component. These specifications are then used to analyze the overall design architecture. In this approach, a design is modeled as a state transition system with a finite number of states and a set of transitions. In addition, a support language is provided to specify a realistic yet simple fault models. Furthermore, a tool support is provided to automatically compose the fault models into the nominal system models for safety analysis.

6.3 Methodology

The proposed framework relies on constructing a finite model of a design and checking it against its desired safety properties. The design model is in essence a finite-state machine, and the fact that it is finite makes it possible to execute an exhaustive state-space exploration to prove that the design satisfies its requirements. Since there is an exponential relationship between the number of states in the model and number of components that make up the system, the *compositional reasoning* approach is used to handle the large state-space problem. The compositional reasoning technique decomposes the safety properties of the system into local properties of its components. These local properties are subsequently verified for each component. The combination of these simpler and more specific verifications guarantees the satisfaction of the global safety of the overall system architecture design. It is important to note that, the safety requirements of the components are satisfied only when explicit assumptions are made on their environment. Therefore an *assume-guarantee* [38, 39, 40, 41, 42] approach is utilized to model each component with regards to its interaction with its environment, i.e, the rest of the system and outside world.

Next in the proposed framework, each design is converted to system-level graph representation. These graph representations are then is used as a tool to convert each design into an adjacency matrix of nodes (components) and edge connections. Subsequently, Non-Linear Dynamical System (NLDS) and epidemic spreading algorithms are used to analyze the propagation of failure in complex engineered system design.

The proposed framework integrates safety by planning and anticipating for unexpected failures and disruptions. From this perspective, safety is considered a dynamic feature of the system that requires constant reinforcement and support on an ongoing basis. It is important to recognize that safety is a feature that results from what a system *does*, rather than a characteristic that system *has*. Therefore, the proof of safety is only conveyed by the absence of failures and accidents. For this reason, the safety-proofing a system design is never absolute or complete. However, in this

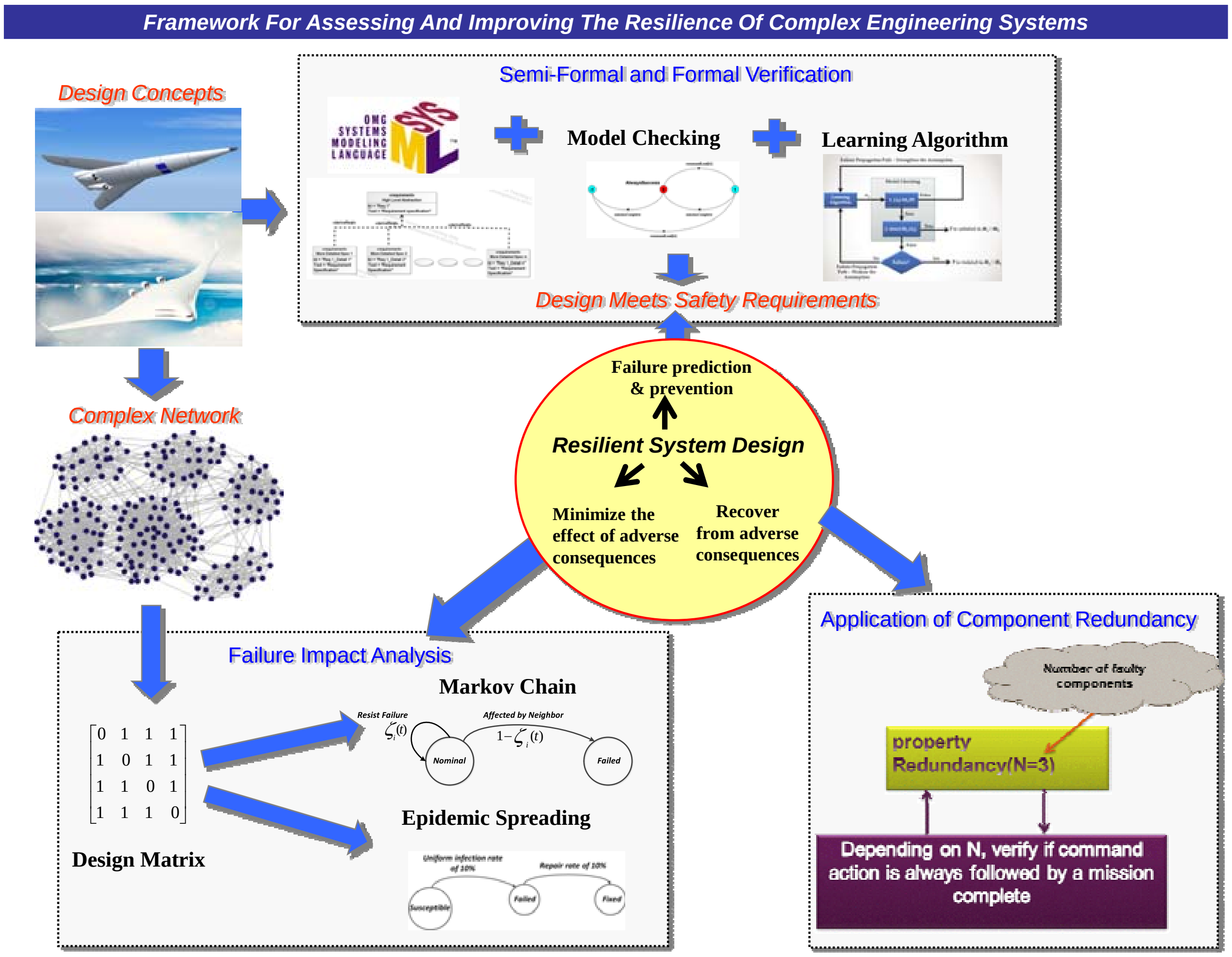

Framework For Assessing And Improving The Resilience Of Complex Engineering Systems
Design Concepts
Semi-Formal and Formal Verification
Model Checking
Learning Algorithm
Design Meets Safety Requirements
Complex Network
Failure prediction & prevention
Resilient System Design
Minimize the effect of adverse consequences
Recover from adverse consequences
Failure Impact Analysis
Markov Chain
Resist Failure
Affected by Neighbor
Nominal
Failed
Epidemic Spreading
Uniform infection rate of 10%
Repair rate of 10%
Susceptible
Failed
Fixed
Design Matrix
Application of Component Redundancy
Number of faulty components
property Redundancy(N=3)
Depending on N, verify if command action is always followed by a mission complete

Figure 6.1: An Overview of The Proposed Approach.

research, the proposed framework biases the odds in the direction that ensures safe system operation by 1- Predicting and preventing adverse consequence 2- Minimizing the adverse consequences, and 3- Recovering from adverse consequences. Fig. 6.1 represents and overview of the approach which will be discussed in detail in the next section.

6.3.1 System Design Verification

We follow a modular and incremental approach for behavioral modeling and verification of a complex system design. The design architecture is used to organize a system in terms of its subsystems, components, and their interactions. Consequently, the general structure of a system is a graph in which its leaves are *primitive components* and the rest are *composite components*. A primitive component describes *behavior* instead of *structure* and has no substructure. On the other hand, a composite component inherits its behavior from its sub-components.

The behavioral modeling requirements are motivated by the quad-redundant Electro-Mechanical Actuator (EMA) [43] of a Flight Control Surface (FCS) of an aircraft which is developed in a program sponsored by NASA. The positions of the FCSs are usually controlled using a flight control surface actuation system. The FCS actuation system responds to position commands sent from the flight crew to move the aircraft FCS to the command positions. In this case study, this movement is effected via quad-redundant actuators (Fig. 6.2) that are coupled to the flight control surfaces. The EMAs are arranged in a parallel fashion; therefore, each actuator is required to tolerate a fraction of the overall load.

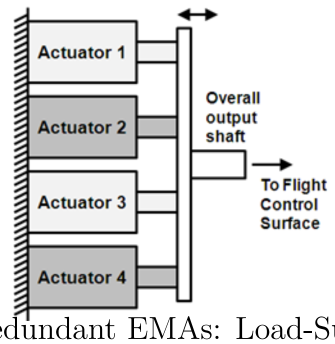


Figure 6.2: Quad-Redundant EMAs: Load-Summing/Parallel Mode.

6.3.1.1 Modeling Behavior

In order to transform the requirements and the design architecture presented in the previous section into a finite model we use finite labelled transition systems (LTS). The LTS model is expressed graphically by its alphabet, transition relation, and states including single initial state. Labelled Transition Systems (LTS) such as $\boldsymbol{T}$ is defined as:

$\mathbf{T} = (S, L, \rightarrow, s_0)$.

A set **S** of **states**

A set **L** of **actions**

A set $\rightarrow$ of transitions from one state to another.

An initial state $s_0 \in S$

As an example consider the following LTS model of a *command unit subsystem* of the quad-redundant EMAs:

{*commandLoad[1..4]*,{*missionComplete,resetShaft,timeout*}}

Fig. 6.3 represents this model graphically. State *0* corresponds to the command unit resetting the output shaft of the FCS before sending any load command to the EMAs. By performing the action <*commandLoad[1..4]*>, the command unit requests a range of load values between *1* to *4* from EMAs. Then, the command unit expects two possible responses <{*timeout and missionComplete*}>. The <*mission is completed*> event occurs when EMA(s) have maintained the specified load to the FCS throughout the mission. On the other hand, the <*timeout*> event occurs in the case where all four EMAs have failed to provide the required load.

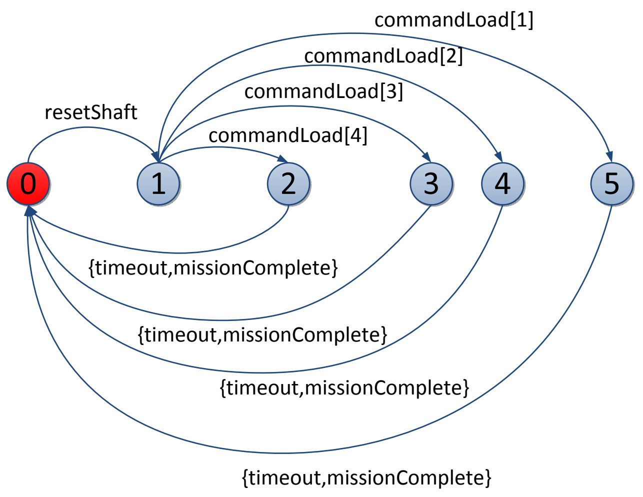

commandLoad[1]
commandLoad[2]
commandLoad[3]
commandLoad[4]
resetShaft
0
1
2
3
4
5
{timeout,missionComplete}
{timeout,missionComplete}
{timeout,missionComplete}
{timeout,missionComplete}

Figure 6.3: LTS Model of the Command Unit Subsystem.

Table 6.1: Syntax and Informal Semantics Of The FSP

FSP Notation		
$\boldsymbol{a \rightarrow P}$	Action prefix	Action $\boldsymbol{a}$ behaves as described by $\boldsymbol{P}$
$\boldsymbol{a \rightarrow \ P \vert b \rightarrow Q}$	Choice	After an initial action is performed subsequent behavior is described by $\boldsymbol{P}$ if the first event was $\boldsymbol{a}$ *or* by $\boldsymbol{Q}$ if the event was $\boldsymbol{b}$
$\boldsymbol{P \parallel Q}$	Parallel Composition	Composition of $\boldsymbol{P}$ and $\boldsymbol{Q}$ where composed LTSs interact by synchronizing on common actions shared in their alphabets with interleaving of the remaining actions.
/	Relabelling	Change the names of action labels. This is usually done to ensure that composed processes synchronizes on the correct actions.

This type of graphical modeling, however, could easily become unmanageable for large complex systems. Therefore, an algebraic notation known as Finite State Process (FSP) [44] is used to define the behavior of processes in a design. FSP is a specification language as oppose to modeling language with semantics defined in terms of LTSs. The FSP basic operators are represented in Table 6.1. Every FSP model has a corresponding LTS description and vice versa. The LTS *CommandUnit* represented in Fig. 6.3 can be expressed in FSP as shown in Table 6.2.

Table 6.2: FSP Description of Command Unit

FSP Notation
1: **CommandUnit = (resetShaft → commandLoad[L] → {timeout,**
2: **missionComplete} → CommandUnit).**

After modeling the *commandUnit* the next primitive component to be modeled is the *controller* subsystem. The controller gets the load command from the command unit and actively regulates the current to each EMA at every time step. The difference between the external load and the total actuator load response is used to accelerate or decelerate the output shaft. If the controller perceives that the output shaft position response is falling behind the commanded position, it will increase the current flow to the EMAs. As depicted in Table 6.3, in the FSP description of the controller, a repetitive behavior is defined using a recursion. In this context recursion is recognized as a behavior of a process that is defined in terms of itself, in order to express repetition.

Table 6.3: FSP Description of Controller

FSP Notation
1: **Controller = (getLoad[l:L] → Controller[l]),**
2: **Controller[t:L] = (timeout → Controller**
3: **| sendLoad→allLoadsCompleted→getShaftPosition[x:Positions]**
4: **→if (x ≥ t) then (missionComplete→Controller)**
5: **else Controller[t]).**

The partial LTS model of the controller is depicted in Fig. 6.4. The *controller* performs action <*getLoad*[*l*..4]>, and then behaves as described by <*Controller*[*l*]>. *Controller*[*l*] is a process whose behavior offers a choice, expressed by the choice operator "|". *Controller*[*l*] initially engages in either <*timeout*> or <*SendLoad*>. The action <*timeout*> is performed when all actuators fail, otherwise <*SendLoad*> is utilized. Subsequently, after sending the required load to each EMA, feedback signals are sent to inform the controller of completion of tasks by labeling the action with <*all Loads Completed*>. This results in the controller to perform the action <*get Shaft Position*>. At this stage, the controller compares the new position with the required shaft position, if the shaft has reached the required position then the <*mission is completed*>. Otherwise, the behavior is repeated until the shaft reaches the required position.

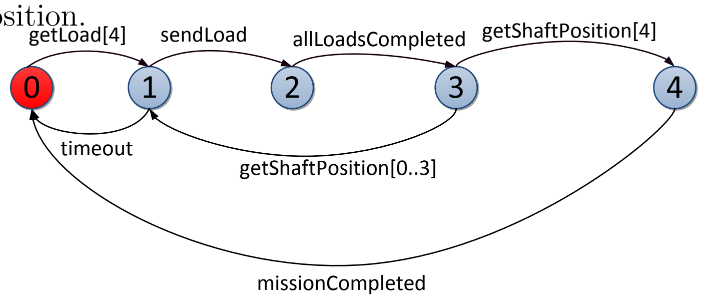


Figure 6.4: LTS Model of the Controller Subsystem.

Next in the modeling process is the *Electro Mechanical Actuator* unit, which receives the load command from the controller and carries out the operation. The Electro Mechanical Actuator is modeled in Table 6.4 with *Jammed* and *Disengaged* as part of its definition. If during the time of maintaining the specified torque or load the EMA functions according to specification, the signal <*"all loads are completed"*> is sent to the controller. Otherwise, the EMA is considered non-operational or jammed. In the jammmed mode, the EMA is incapable of maintaining the required load and prevents the rest of the EMAs from moving. Therefore, it needs to be disengaged from the system. Fig. 6.5 presents the LTSA model of the EMA subsystem including the failure mode of <*jam*> in state (3) and its following consequences.

The next subsystem to be modeled is the *Diagnostic Block*, which performs actuator health assessments, and makes decisions on whether or not to disengage any faulty actuators from the flight control surface. Each time an EMA is in its fail-

Table 6.4: FSP Description of EMA

FSP Notation
1: **EMA = (recLoad → performLoad → (allLoadsCompleted → EMA**
2: | **jam → block → Jammed)),**
3: **Jammed = (recLoad → Jammed**
4: | **disengage → unblock → Disengaged),**
5: **Disengaged = (recLoad, allLoadsCompleted, timeout → Disengaged).**

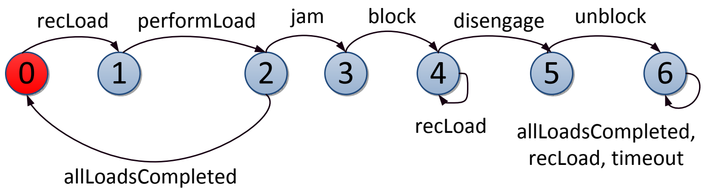


Figure 6.5: LTS Model of the EMA Subsystem.

ure mode of <*jam*>, the diagnostic block performs the <*disengage*> action and disengages the faulty EMAs from the rest of the system. Table 6.5 represents the FSP model of the diagnostics subsystem and Fig. 6.6 depicts the corresponding LTS model of the diagnostics subsystem.

Table 6.5: FSP Model of Diagnostics for The Redundant EMAs

FSP Notation
1: **Diagnostics** = [e : **EMAs**]**.jam** → [e]**.disengage** → **Diagnostics**

Finally, the last subsystem to be modeled is the *shaft*. The overall shaft dynamics, which represent the motion of all of the four EMAs, are represented by a single block. As illustrated in Table 6.6, the shaft model keeps track of the *position* of the output shaft while each EMA performs a load. At each simulation step the current position of the output shaft is compared with the command position to make sure that the required position is reached by the output shaft. The $Shaft[0..4]$ is a process whose behavior offers a choice of <*load, block, positionIs, or reset*>. The *block* action occurs

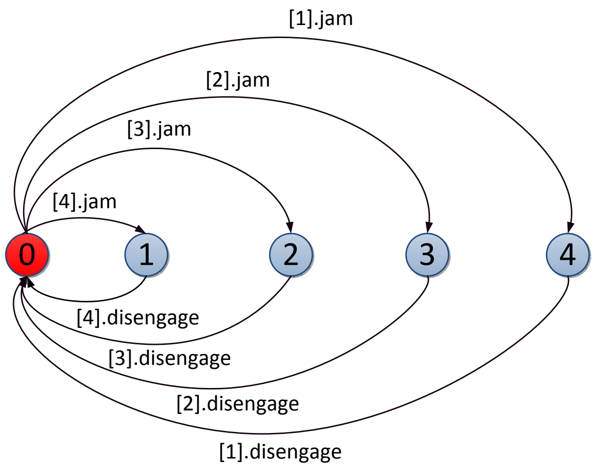


Figure 6.6: LTS Model of The Diagnostics Subsystem.

when one of the EMAs is in its failure mode and causes the shaft to become non-operational or <*Blocked*>. On the other hand, the <*positionIs*> action is utilized when the controller requires to receive information about the current position of the shaft. The <*reset*> action occurs when the command unit sends the reset signals to the shaft.

The *Blocked* process in the *shaft* subsystem represents the failure mode of the shaft. Whenever the diagnostic performs a <*disengage*> operation on an EMA, the <*unblock*> action is performed.

Table 6.6: FSP Description of The Shaft

FSP Notation
1: **Shaft = Shaft[0],**
2: **Shaft[p: Positions] = (load → if (p < MAXPosition) then Shaft[p+1]**
3: **else Shaft[p]**
4: **| block → Blocked[p][1]**
5: **| positionIs[p] → Shaft[p]**
6: **| reset → Shaft),**
7: **Blocked[p: Positions][b: EMAs] = (load → Blocked[p][b]**
8: **| when (b > 1) unblock → Blocked[p][b-1]**
9: **| when (b == 1) unblock → Shaft[p]**
10: **| when (b < M) block → Blocked[p][b+1]).**

6.3.1.2 Composition

In this research it is assumed that the design is described by a *composition* expression. In the context of system design engineering, the term composition is similar as coupled model. Coupled model, defines how to couple several component models together to form a new model, similarly, composition groups together individual state machines. Such an expression is called a parallel composition, denoted by "‖". The "‖" is a binary operator that accepts two LTSs as an input argument. In the joint behavior of the two LTSs, the transition can be performed by any of the LTS if the action that labels the transition is not shared with the other LTS. Shared actions have to be performed concurrently. Table 6.7 depicts the FSP of the joint behavior of *EMA* and *controller*. The composed LTS model of the two subsystems consists of 161 states and 62 transitions. The shared action between the two models is the <*sendLoad*> action from the controller and the <*recLoad*> action from the EMA, therefore, these two are required to be performed synchronously. In order to change action labels of an LTS, the *relabeling* operator "/" is used, e.g., { recLoad / sendLoad }.

Table 6.7: Parallel Composition of EMA (Table 6.4) and Controller (Table 6.3)

FSP Notation
1: ‖ **Leg = (EMA ‖ Controller) / { recLoad / sendLoad }.**

As a result, the composed model consists of the following actions:

{{*allLoadsCompleted, block, disengage*},

getLoad[1..4], getShaftPosition[0..4],

{*jam,missionComplete,performLoad,*

recLoad,timeout,unblock}}

As described, composed LTSs interact by synchronizing on common actions shared in their FSP models with interleaving of the remaining actions. Also, it is important to note that the parallel composition operator enables both associative and commutative composition; therefore the order of LTS models that are composed together is insignificant, e.g., ||Leg=(Controller || EMA). Table 6.8 presents some of the state

Table 6.8: Leg Subsystem: Two Possible Transitions

EMA: Nominal Mode	EMA: Failure Mode
1: ctrl_getLoad.2	*1*: ctrl_getLoad.2
2: EMA_recLoad	*2*: EMA_recLoad
3: EMA_performLoad	*3*: EMA_performLoad
4: LoadsCompleted	*3*: **EMA_jam**
5: ShaftPositionIs.1	*4*: **Shaft_block**
6: EMA_recLoad	*5*: **EMA_Disengage**
7: EMA_performLoad	*6*: **Shaft_Unblock**
8: LoadsCompleted	*7*: LoadsCompleted
9: getShaftPosition.2	*8*: ShaftPositionIs.1
10: EMA_performLoad	*9*: **timeout**
11: missionComplete	–

transitions (or sequence of actions) produced by the composed model. Two possible executions under the EMA's nominal and faulty conditions are considered. In nominal mode, the EMA receives a request from a controller to provide two unit loads. At each time step, EMA performs one unit load and repeats until the output shaft reaches the required position of two that is when the <*missionComplete*> actions is performed. In the failed mode, initial actions are the same as nominal mode until an EMA jams. The jammed EMA blocks the rest of the system from moving until it is disengaged. The process is followed by the <*Unblock*> action which unblocks the shaft allowing the rest of the system to be freed. By this time, the EMA has provided one unit load before being disconnected from the rest of the system. Since, the <$ShaftPositionIS$> shows the current position of the shaft being one instead of two, the EMA is required to perform one more unit of load. However, the disengaged EMA is incapable of doing so resulting in a <*timeout*>. The <*timeout*> occurs only when there are no EMAs to perform the required load.

Another example of a compositional model is the composition of the shaft model. Composition implements the motion of all four shafts that are connected on one side to the four EMAs, and on the other side to the output shaft of the Flight Control Surface (FCS). As illustrated in Table 6.9, the actions of the *shaft* are relabeled so that the <*load*> becomes <[1..4].*load*>. This results in identification of actions for each EMA, e.g., <[1].*load*> (EMA#1 is loading) or <[2].*block*> (EMA#2 is blocking).

Table 6.9: FSP Description of The Redundant Shaft

FSP Notation
1: ‖ **RedShaft = Shaft / { [e:EMAs].load / load,**
2: **[e:EMAs].block / block,**
3: **[e:EMAs].unblock / unblock}.**

The next hierarchy of the compositional model is based on the composition of the two compositional models of the *leg* and *redundant shaft* and two primary components of *diagnostic*, and *command unit* (Table 6.10). This compositional approach reflects an organization of the design components in a hierarchical structure.

So far, we provided the basis for decomposing and modeling the system based on the modular description of the design components and subsystems. In the next section, the process of expressing the desired safety properties in terms of a state machine or LTS is described. The advantage is that both the design and its requirements are modeled in a syntactically uniform fashion. Therefore, the design can be

Table 6.10: FSP Description of Quad-Redundant EMA System

FSP Notation
1: **‖RedSystem= (shaft:RedShaft ‖ d:Diagnostics**
2: **‖leg[EMAs]:Leg‖commandUnit)**
3: **/{shaft.positionIs / leg[EMAs].getShaftPosition,**
4: **commandLoad / leg[EMAs].getLoad,**
5: **d[i:EMAs].jam / leg[i].jam,**
6: **leg[i:EMAs].disengage / d[i].disengage,**
7: **shaft[i:EMAs].block / leg[i].block,**
8: **shaft[i:EMAs].unblock / leg[i].unblock,**
9: **shaft[i:EMAs].load / leg[i].performLoad,**
10: **legsRecLoad / leg[EMAs].recLoad,**
11: **allLoadsCompleted / leg[EMAs].allLoadsCompleted,**
12: **missionComplete / leg[EMAs].missionComplete,**
13: **timeout / leg[EMAs].timeout**.}

compared to the requirements to determine whether its behavior conforms to that of the specifications.

6.3.1.3 Safety LTS And Safety Property

In the context of this work, the properties of a system are modeled as safety LTSs. A ***safety*** LTS contains no failure states. In modeling and reasoning about complex systems, it is more efficient to define safety properties by directly declaring the desired behavior of a system instead of stating the characteristics of a faulty behavior. In a Finite State Process (FSP), the definition of properties is distinguished from those of subsystem and component behaviors with the keyword *property.* For example, the following model is constructed to state the safety requirements of the quad-redundant EMAs.

Table 6.11: FSP Description of The Safety Requirement

FSP Notation
1: **property**
2: **SafeOpn = (commandLoad[t:L] → missionComplete → SafeOpn).**

The <*safeOpn*> property of Table 6.11, expresses the desired system behavior that any <*commandLoad*[1..4]> action eventually shall be followed by a <*missionComplete*> action. As it is depicted in Fig. 6.7, while translating the FSP notation of a prop-

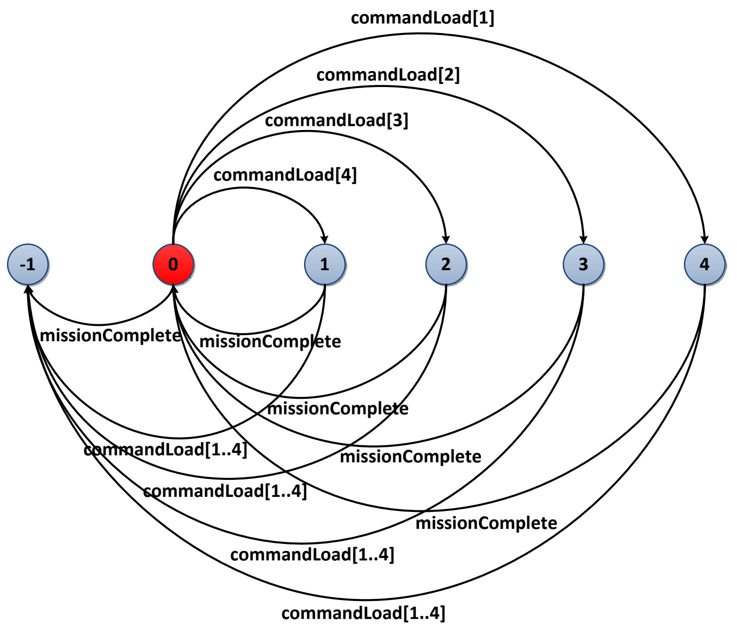


Figure 6.7: The LTS Model Of The Safety Operation Property.

erty, the verification algorithm automatically generates the transitions that violate the properties within the LTS model. For example, at state {0}, the occurrence of <*missionComplete*> without previously performed <*commandLoad*> leads to a failure state. Another example is the consecutive execution of the <*commandLoad*>. The LTS of Fig. 6.7 is recognized as an *error* LTS with the failure state of *-1*.

6.3.2 Reliability Analysis in Complex Networks

Assuming **A** signifies the adjacency matrix of an engineered system under study with n components, A is defined as follows:

$$\mathbf{A}_{ij} = \begin{cases} 1 & \forall\ [(\text{i,j}) \mid (\text{i} = \text{j}) \text{ and } (\text{i,j}) \in \Lambda] \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where Λ symbolizes the set of components. **A** is a square symmetric matrix with diagonal entries of zero.

The graph layout can be determined in any number of different methods. As discussed earlier, the edge connections between components can be defined topologically or by Energy, Material, and/or Signal (EMS) relations. A topologically defined graph has components which are connected with physical justifications. For example, if two components are physically connected together within a design, they are connected within the graph. EMS related connectivity rules describe the energy, material, and/or signal flows between components. Therefore, if two design components share a flow variable (or a flow relationship), they would be *"connected"* within the graph and represented with a **"1"** within an adjacency matrix.

6.3.2.1 Non-Linear Dynamical System (NLDS) Modeling

The NLDS propagation model provides an indication for the length of time to full propagation according to the graph layout defined by an adjacency matrix. In the proposed model, a universal failure cascading rate β ***(0*** $\leq$ β $\leq$ ***1)*** for each edge

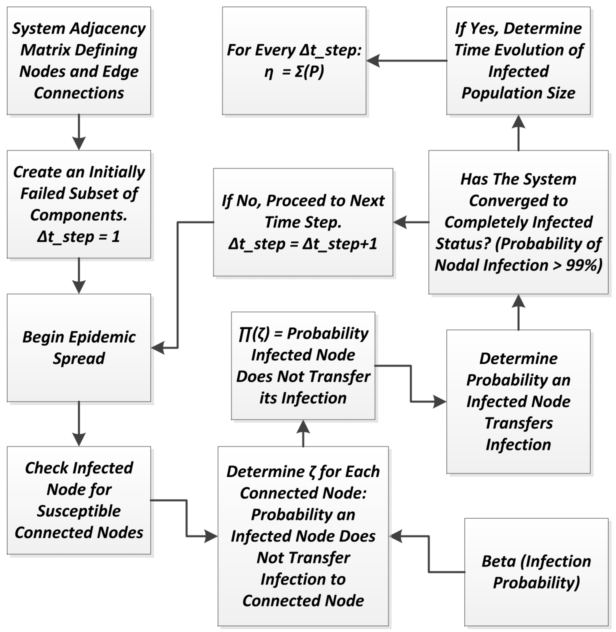

System Adjacency Matrix Defining Nodes and Edge Connections
Create an Initially Failed Subset of Components. Δt_step = 1
Begin Epidemic Spread
Check Infected Node for Susceptible Connected Nodes
Determine ζ for Each Connected Node: Probability an Infected Node Does Not Transfer Infection to Connected Node
Beta (Infection Probability)
∏(ζ) = Probability Infected Node Does Not Transfer its Infection
Determine Probability an Infected Node Transfers Infection
Has The System Converged to Completely Infected Status? (Probability of Nodal Infection > 99%)
If No, Proceed to Next Time Step. Δt_step = Δt_step+1
If Yes, Determine Time Evolution of Infected Population Size
For Every Δt_step: η = Σ(P)

Figure 6.8: Overview of Non-Linear Dynamical System Model.

connected to a faulty component is defined. The model is based on discrete time-steps $\Delta \boldsymbol{t}$, with $\Delta \boldsymbol{t} \rightarrow \boldsymbol{0}$. During each time interval $\Delta \boldsymbol{t}$, a faulty component i infects its neighboring components with probability β. Model3.pdf

Fig. 6.8 illustrates an overview of the step required to analyze the propagation of failure in complex engineered system design using the Non-Linear Dynamical System (NLDS) Model.

6.3.2.2 Epidemic Spreading Model

Fig. 6.9 depicts an overview of the steps required to analyze the propagation of failure in a complex engineered system design using the epidemic spreading model. In this approach, the theoretical model is based on the concept that each component in the

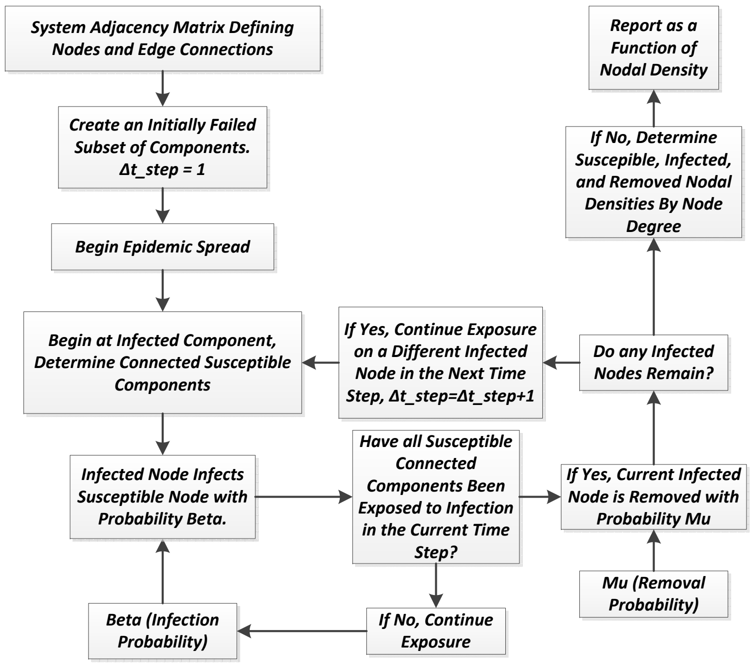

System Adjacency Matrix Defining Nodes and Edge Connections
Create an Initially Failed Subset of Components. Δt_step = 1
Begin Epidemic Spread
Begin at Infected Component, Determine Connected Susceptible Components
Infected Node Infects Susceptible Node with Probability Beta.
Beta (Infection Probability)
Have all Susceptible Connected Components Been Exposed to Infection in the Current Time Step?
If No, Continue Exposure
If Yes, Current Infected Node is Removed with Probability Mu
Mu (Removal Probability)
Do any Infected Nodes Remain?
If Yes, Continue Exposure on a Different Infected Node in the Next Time Step, Δt_step=Δt_step+1
If No, Determine Suscepible, Infected, and Removed Nodal Densities By Node Degree
Report as a Function of Nodal Density

Figure 6.9: Overview of Susceptible - Failed - Fixed (SFF) Model.

complex system design can exist in a discrete set of states. The failure propagation changes the state of a component from nominal to failure or from failure to fixed. As a result, the model is classified as a susceptible - failed - fixed (SFF) model, in which components only exist in one of the three states.

The proposed methodology is based on the universal rate (μ) in which the failed components are fixed in the design, whereas susceptible components are affected by the failure at a rate (λ) equal to the densities of failed and susceptible components. In addition, $\bar{k}$ is defined as a the number of contacts that each component has per unit time. It is important to note that the assumption made in this proposed model is based on the fact that the propagation of failure is proportional to the density of the faulty components.

Two failure propagation models are used to demonstrate the resilience of a quad-redundant Electro Mechanical Actuator (EMA) design against cascading failure. The design is analyzed for its resilience to propagation by evaluating the design for length of time to full propagation (NLDS) and for the breadth of propagation (SFF) when a failure is introduced.

In order to gauge the resilience to propagation by each designed system, an initial set of components in a state of failure is defined so the failure can propagate along the underlying graph structure from these components. For the sake of comparison, each design has been compared twice, once with an initially failed minimally connected component and once with a highly connected component. Specifically, a minimally connected temperature sensor and a highly connected EMA engine are selected as the failure origins.

As it can be seen in Fig. 6.10 the population of the infected components with respect to time is different for the two experiments. In the left hand side of Fig. 6.10, the defect at the origin of failure in EMA engine caused a drastic increase in infected population size. This occurred near the second time step for quad-redundant EMA design. In this case, the failure is able to spread much more quickly because the EMA engine node is the most highly connected component in this design. The result confirms the expectation that a more highly connected component propagating failure to neighboring component more quickly, while a minimally connected component, such as a sensor, results in slower failure propagation.

The NLDS model proves that more highly connected components spread a failure

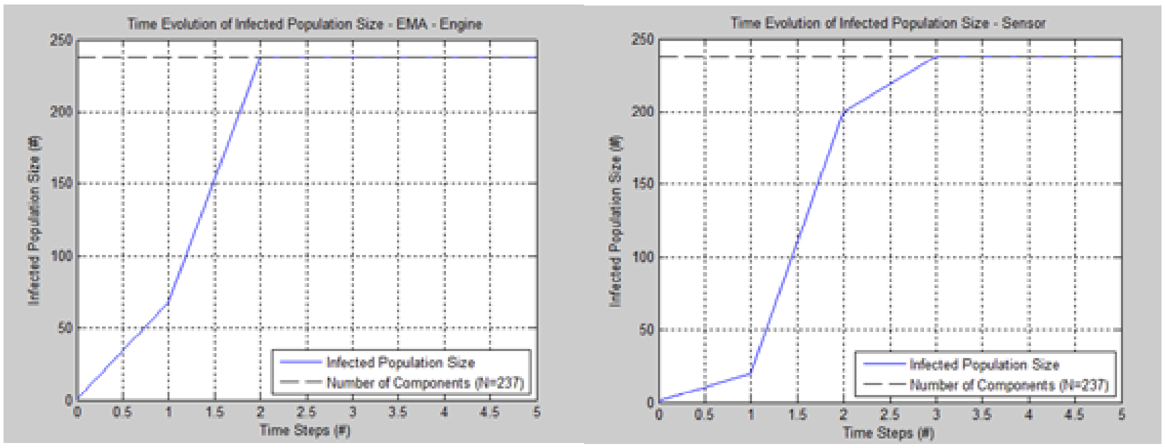


Figure 6.10: Time Evolution of Faulty Components' Population Size (Origin of Failure: (Left Picture: EMA Engine) and (Right Picture: Sensor)).

much faster. Therefore, nodal hubs, or very modular areas of a design are more detrimental to the rapid spread of a failure.

Unlike the NLDS model, the SFF epidemic spreading model is based on the idea that failure propagations can be stopped by fixing the faulty components. The SFF model operates by the spread of a failure from an initially failed component just as the NLDS model does. However, the SFF model is not a probabilistic model that is solely dependent on the architecture of an adjacency matrix as the NLDS model is. Instead, the SFF model requires a time step dependent simulation of the spread of a component failure. As with the NLDS model, a time step is regarded as sufficiently close to zero so that only the current population of failed components transmit a failure.

Each "faulty" component has an opportunity to infect a neighboring susceptible component in the next time step. In one time step, a component infects its connected neighbors according to a uniform failure probability. The simulation run for the SFF model was conducted at $\lambda = 10\%$. After a component has had an opportunity to infect its neighbors, the infected component would then be fixed in the conceptual design to resist the same failure according to the probability of failure removal $\mu = 10\%$. A repaired component is either considered faulty without the ability to transfer the failure to the neighboring components or is susceptible but resistant to the failures of its connected neighbors. Therefore, the cascading failure could be stopped with the provision that enough faulty components become repaired in the design before they are able to fully propagate the failure. That is, propagations can be halted if all transmission routes are blocked by repaired components.

The same designs were used with the epidemic spreading model as were used with the NLDS model. Additionally, the same initial failure conditions were used. A temperature sensor was initially failed as a minimally connected component. An EMA engine node was then initially used to propagate the failure as a highly connected component. Fig. 6.11 shows the epidemic spreading graphs.

In the SFF algorithm, the failure propagation is based on connections, therefore the results must be reported in terms of faulty component density. As it is depicted in Fig. 6.11, each colored data set is representative of a set of components with the same degree e.g. red colored data set represents *62* components in the system design with only three connections. Therefore, each set of components has a failure density ranging from 0 to 1; 0 representing that no components of that degree are infected and a 1 representing every component of that degree being infected. Fixed components are not considered faulty. Consequently, a plot of faulty component density fluctuates intermittently between 0 and 1, however eventually settles at 0 as

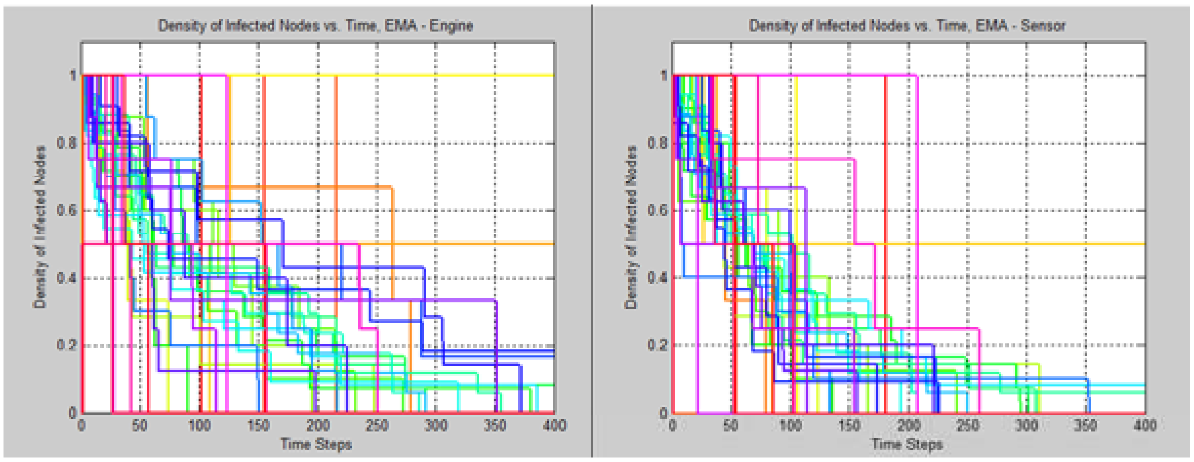


Figure 6.11: Time Evolution of Faulty Components Density for The quad-redundant EMA design (Origin of Failure: (Left Picture: EMA Engine) and (Right Picture: Sensor)).

all failed components are fixed in the system design. When a minimally connected component, such as a temperature sensor, is chosen as a failure origin, it is compared to an initially infected highly connected component, such as an EMA engine, the cascade spreads much slower, as expected.

The left hand side of Fig. 6.11, illustrates the simulation results for an initially failed, highly connected EMA engine. The plots present more immediate increases in infection density, regardless of component degree when a highly connected component is failed initially. However, once the initial infection has passed, and the failure density begins to subside, the infection density is reduced. This is because a stopped failure gets repaired probabilistically according to a uniform rate.

6.3.3 Design Analysis

In order to produce an architecture that can be used to verify all the required design functionalities, the behavioral model of the system is composed with the defined safety properties. For example, the $<SafeOpn>$ property of Fig. 6.7 is composed with the target system of Table 6.10. Then the verification algorithm analyzes all execution paths of the composed model to ensure the specified property holds for all executions of the system. The violation of linear-time safety properties is indicated by finite path fragments that end in a failure state. Therefore, all finite traces of the system are tested to ensure that they satisfy the safety requirements. The system satisfies the safety property if and only if the failure state is not reachable in (SafeOpn || RedundantSystem). When the design satisfies the property, then the composition of SafeOpn || RedundantSystem behaves similar to the behavioral model of the redundant system. Therefore, the composition does not affect the system behavior. However, when a safety property is violated, a failure propagation path consists of series of actions that lead to failure state is generated.

Table 6.12: FSP Model of Safety Property And System

FSP Notation
1: **||CheckSafeOpn = (SafeOpn || RedundantSystem).**

In the case of the $<SafeOpn>$ property, the verification algorithm had detected a property violation. Table. 6.13 represents the sequence of actions that lead to the failure state.

After resetting the output shaft, the command unit sends a request for two units of load. Later in the process, the diagnostics subsystem identifies a jammed actuator causing shaft #1 to be blocked. As it is presented in line #25, the overall position of

Table 6.13: Trace To Property Violation

1: shaft.reset	***15***: shaft.2.unblock
2: commandLoad.2	***16***: d.3.jam
3: legsRecLoad	***17***: shaft.3.block
4: shaft.1.load	***18***: leg.3.disengage
5: d.1.jam	***19***: shaft.3.unblock
6: shaft.1.block	***20***: d.4.jam
7: shaft.2.load	***21***: shaft.4.block
8: shaft.3.load	***22***: leg.4.disengage
9: shaft.4.load	***23***: shaft.4.unblock
10: leg.1.disengage	***24***: allLoadsCompleted
11: shaft.1.unblock	***25***: shaft.positionIs.1
12: d.2.jam	***26***: timeout
13: shaft.2.block	***27***: shaft.reset
14: leg.2.disengage	***28***: commandLoad.1

the output shaft connected to the Flight Control Surface (FCS) is reported one. After the load provided by shaft #1, the loads from other three shafts are not performed due to the fact that shaft #1 has blocked the system. After disengaging leg #1, the system returns to the operational mode. However, at this point the diagnostic block detects that the remaining actuators have also failed causing a <*timeout*> to occur. The second load command is sent by the controller to reach the required position of two, yet, sending the second load command before a <$missionComplete$> results in violation of <$SafeOpn$> property.

It can be concluded from the result of the verification that the failure of all four EMAs leads to catastrophic system failure. However, it is possible to extend the LTS model of <$SafeOpn$> to constrain the number of failures in a way that the system never reaches the catastrophic failure. For example, in the case of quad-redundant EMAs, the system can tolerate up to three failures without reaching the catastrophic failure condition. This way, system designers can impose a new requirement of the form "*if up to N number of EMAs fail then the catastrophic failure condition shall* *not occur.*". To achieve this, the following generic (or parameterized) safety property with the following constants and a range definitions is used:

- const N =4 \\ *number of faulty EMAs* [1]
- const M =4 \\ *number of EMAs*
- range EMAs = 1..M \\ *EMA identities*

In order to prevent the system from reaching the catastrophic event of <*timeout*>, it is essential to complete the mission and provide the required loads based on the command signal. Therefore, the events of interest are the sent command signal, the jammed actuators, and the completion of the mission. Consequently, as depicted in the LTS model of Fig. 6.12 <$Fault\_Tolerance$> property contains {<commandLoad[1..4],d[1..4].jam, and missionComplete>} actions. The property of Table 6.14, maintains a count of faulty EMAs with the variable f. To model the fact that every command signal must be followed by a <*missioncomplete*>, the processes in line #3 and 8 are required to constrain the number of faulty EMAs (f) to a number defined by the parameter of the property (N).

Table 6.14: FSP Model of Fault Tolerance Property

```
1: property
2: Fault_Tolerance(N=4) = Jammed[0],
3: Jammed[f: 0..M] =(when(f ≤ N)commandLoad[L] → CompleteMission[f]
4:             |when (f>N) commandLoad[L] → Jammed[f]
5:             |d[EMAs].jam → Jammed[f+1]
6:             |missionComplete → Jammed[f]),
7: CompleteMission[f:0..M] = (missionComplete → Jammed[f]
8:             |when (f<N ) d[EMAs].jam → CompleteMission[f+1]
9:             |when (f==N) d[EMAs].jam → Jammed[f+1]).
```

[1] by default is set to *4* but it can be redefined during the instantiation process.

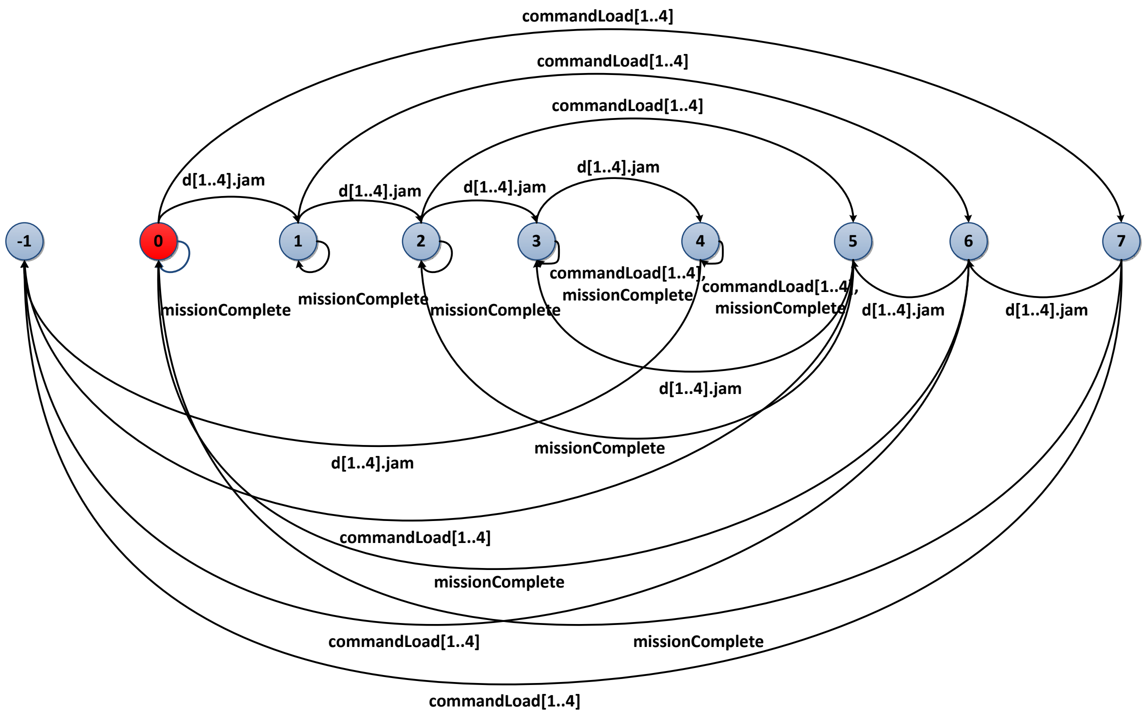


Figure 6.12: LTS Model Of The Fault Tolerance Property.

As can be seen in the compositional model of Table 6.15, the $<Fault\_Tolerance>$ property is predefined with $N = 2$. Therefore, permitting only two out of four EMAs to fail during the system operation. Safety analysis using the LTS analyzer verifies that the safety property is satisfied. The composed LTS model of Table 6.15 consists of 2^{42} states, however the verification algorithm reduced the number of states to *10733*. The same result is obtained with three EMAs failing. However, when

Table 6.15: Compositional Model Of The System And Safety Property

```
1: ||Extend_CommandUnit=(Fault_Tolerance(2)||CommandUnit)
2:         /shaft.reset/resetShaft.
3: ||Check_Property =(Extend_CommandUnit||RedundantSystem).
```

the property is instantiated allowing four EMAs to fail, the safety analysis verifies that the property is violated and a failure propagation path similar to the one in

Table 6.13 is produced. Therefore, the generic safety property modeled in Table 6.14 verifies that the system never reaches the failure condition of *total loss* if and only if $N \leq M\text{-}1$ where N is the number of faulty EMAs and M is the total number of EMAs.

From the result of case study: the characterization of the system architecture by its subsystems and components, the FSP annotation of the failure behavior of each of them, and the system level safety analysis based on components' interaction lead to achieving a manageable verification procedure. As compositional reasoning approach significantly reduces the number states to be explored, exhaustive checking of the entire state space is made feasible. This is especially important where the exhaustive simulation is too expensive and non-exhaustive simulation can miss the critical safety violation. Furthermore, this type of safety analysis are very helpful during the early stages of the design because they provide the required information to implement the appropriate level of component redundancies. It is important to note that, even though the application of component redundancy improves system reliability, it also adds cost, weight, size, and higher power consumption.

6.4 Conclusion and Future Work

This paper presented a new combination of compositional verification and model checking based approach for system design safety analysis. The aim of compositional reasoning is to improve scalability of the design verification process by decomposing the original verification task into subproblems. The simplification is based on the assumption that a design can be described by a composition expression that groups together individual state machines, while its requirements are also modeled in a syntactically uniform fashion. Then the system model is compared to its safety requirements and design specifications to verify that its behavior conforms to that of the specifications. The proposed methodology is verified to be effective for the safety assessment of the early definition of an engineered system design. It offers a uniform platform for both design engineers to verify a system design and for safety analysts to automate specific parts of safety assessment process. The major benefit of the proposed approach is in its ability to tightly integrate safety and design activities. The main functionalities provided by the approach include automatic failure injection based on a database of predefined failure modes, e.g., automatic failure of actuators one after another, automatic generation of fault trees, and exhaustive safety property verification with the help of model checking algorithms.

In addition, we have illustrated modeling and verification using generic safety properties. The approach can be generalized to any relative complex system design component where redundancy of similar components indicate an opportunity for parameterized reuse of requirement models in the verification process. As it is illustrated, the generic $<Fault\_Tolerance(N)>$ property constituted a configurable process of safety verification analysis to determine whether a failure has occurred. This is specifically important for verifying the safety properties of dynamic engineered systems, where the component's failure characteristics are dependent on the interaction with the rest of the system and the outside world. The generic safety property formulation accommodates this ongoing design requirements change process.

In future work, we intend to take into consideration a specified mission time and the average life which is the estimated time until component or system fail. Also, the more complex redundancy problem will be analyzed within a larger design model where the scalability of the system can be validated. In addition, it is our goal to investigate different aspects of fault tolerant system design requirements while taking into account automatic injection of multiple failures and reasoning about different types of recovery strategies.

Chapter 7: Epilogue

In this research, resilience is addressed from different viewpoints and a framework is presented that enables the design of resilient systems. Resilience design is considered an ability to design a system that is able to predict and prevent failure through exhaustive verification and by using appropriate learning algorithms. Secondly, resilience design is characterized as part of a physical infrastructure of the design that is robust and has the ability to survive disruptions. Lastly, through the resilience design approach, systems are designed to recover from disruptions by attempting to return to the pre-disruption state.

In addition, this research clarifies the difference between safety and resilience. Safety is characterized as an emerging behavior of the system that results from interactions among system components and subsystems, including software and humans. This is where designing a resilient system plays a crucial role in developing a proactive design practice for exploiting insights on faults in complex systems. In this context, system failure is viewed as an inability of the system to adapt and recover from disruptions, rather than components' and subsystems' breakdown or malfunction.

Importantly, this work presented a framework for assessing and improving the resilience of complex systems during the early design process. The framework comprises failure prediction and prevention techniques, analysis the effect of design topology on the propagation of failures, and provides methodologies for system recovery from disruptions. The reason for these layers of analyzes is to provide the system designers with a set of tools to support them to integrate safety and resilience where it is needed.

This work presented a compositional verification approach and its novel application in the area of system design and verification through pre-verification of system components and compositional reasoning. The aim of compositional reasoning is to improve scalability of the design verification problem by decomposing the original verification task into subproblems. Also, two propagation models, a Non-Linear Dynamical System (NLDS) model and an epidemic spreading model, are studied to be used during the early design of complex systems. From the two models, equations are provided to model the propagation characteristics of failures in complex engineered systems. The NLDS propagation model provides an indication for the length of time to full propagation according to a graph layout. The Susceptible-Failed-Fixed (SFF) epidemic spreading model provides an indication of the extent of a cascade according to a graph layout.

Future work will extend the existing verification technique to include systems that exhibit probabilistic behavior. The approach will be based on the multi-objective probabilistic model checking. Properties of these models are formally defined as probabilistic safety properties. Furthermore, the behavioral modeling and the automata learning algorithms require modification to support non-deterministic systems. It is also necessary to explore symbolic implementation of the algorithms for increased scalability.

In addition this work presented a new technique for analysis of failure propagation in early design based on the design architecture. Future work will analyze the effect on a complex engineered system if system graph were created using different, and perhaps more complicated justifications. Such justifications could include expanded physical connections that are inclusive of secondary component interactions. They do not necessarily have a physical connection interface and may be produced as a consequence of system operation. This could include heat, noise, and vibration related interactions, among others. In addition, justifications related to energy, material, and signal (EMS) flows would be a necessary next addition to analyze these models for applicability with complex engineered systems. EMS flow relations would create connections between components that share a flow. For instance, two components would be connected as a part of a thermodynamic process if the same working fluid travels from one component to the other. Analyzing multiple adjacency matrices created with various connection justifications per design would provide a more complete look at a designŠs resilience to propagations. This would add computational expense in the analysis of the design and designer effort to create the matrices. An automated method to create the adjacency matrices would solve this issue; however, this would require an interface between a design tool such as a CAD suite and a matrix creator. In order to validate the use of these models, mode and effects analysis and functional criticality level of components will be taken into the consideration and with the use of of an engineered system that is operating in a state of failure should be analyzed by means of Bayes Decision Function to assess risk. By analyzing the time dependent response of a system under failure, the propagation properties predicted by these models can be verified.

References

Chapter 1

[1] Robert G Gregory, Scott J Marcellino, and Seth A Moyer. Analysis of nasa's post-challenger response and relationship to the columbia accident and investigation. Technical report, DTIC Document, 2006.

[2] Dan Braha, Nam Suh, Steven Eppinger, Michael Caramanis, and Dan Frey. *Complex engineered systems*. Springer, 2006.

[3] Jean Pariès. Complexity, emergence, resilience. *Hollnagel, E., Woods, dd, Leveson, N.(editors). Resilience Engineering. Concepts and Precepts. Ashgate*, 2006.

[4] Bernard Pavard, Julie Dugdale, N Bellamine-Ben Saoud, Sandrine Darcy, and Pascal Salembier. Design of robust socio-technical systems. *Resilience engineering, Juan les Pins, France*, 2006.

[5] Nancy G Leveson. System safety engineering: Back to the future. *Massachusetts Institute of Technology*, 2002.

[6] Nancy Leveson. A new accident model for engineering safer systems. *Safety science*, 42(4):237–270, 2004.

[7] AM Madni. Designing for resilience. *ISTI Lecture Notes on Advanced Topics in Systems Engineering*, 2007.

[8] Giles Hutchins. *The Nature of Business: Redesign for Resilience.* New Society Publishers, 2013.

[9] Suniya S Luthar, Dante Cicchetti, and Bronwyn Becker. The construct of resilience: A critical evaluation and guidelines for future work. *Child development*, 71(3):543–562, 2000.

[10] Suniya S Luthar, Dante Cicchetti, and Bronwyn Becker. The construct of resilience: A critical evaluation and guidelines for future work. *Child development*, 71(3):543–562, 2000.

[11] Benjamin S Blanchard, Wolter J Fabrycky, and Wolter J Fabrycky. *Systems engineering and analysis*, volume 5. Prentice Hall Englewood Cliffs, New Jersey, 2010.

[12] Erik Hollnagel, David D Woods, and Nancy Leveson. *Resilience engineering: Concepts and precepts.* Ashgate Publishing, Ltd., 2007.

Chapter 2

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

[2] S. McIlraith, G. Biswas, D. Clancy, and V. Gupta. Hybrid Systems Diagnosis. *in Hybrid Systems: Computation and Control. ser. Lecture Notes in Computer Science, N. Lynch and B. Krogh, Eds.*, 1790:282–295, 2000.

[3] X. Koutsoukos, J. Kurien, and F. Zhao. Monitoring and Diagnosis of Hybrid Systems Using Particle Filtering Methods. *International Symposium on Mathematical Theory of Networks and Systems*, 2002.

[4] M. Hofbaur and B.C. Williams. Mode Estimation of Probabilistic Hybrid Systems. *in Hybrid Systems: Computation and Control. HSCC*, 2289:253–266, 2002.

[5] Marko Čepin. Reliability block diagram. In *Assessment of Power System Reliability*, pages 119–123. Springer, 2011.

[6] D Do. Procedure for performing a failure mode, effects and criticality analysis. 1980.

[7] Robert B Stone, Irern Y Tumer, and Michael Van Wie. The Function-Failure Design Method. *Journal of Mechanical Design(Transactions of the ASME)*, 127(3):397–407, 2005.

[8] Irem Y Tumer, Robert B Stone, and David G Bell. Requirements for a Failure Mode Taxonomy for Use in Conceptual Design. In *Proceedings of the International Conference on Engineering Design, ICED*, volume 3. Paper 1612, Stockholm,, Sweden, 2003.

[9] Daniel Krus and Katie Grantham Lough. Applying Function-Based Failure Propagation in Conceptual Design. ASME, 2007.

[10] Katie Grantham Lough, Robert B Stone, and Irem Tumer. The Risk in Early Design (RED) Method: Likelihood and Consequence Formulations. In *2006 ASME International Design Engineering Technical Conferences and Computers*

and Information In Engineering Conference, DETC2006, September 10, 2006-September 13, 2006.

[11] Erich Devendorf and Kemper Lewis. Planning on mistakes: An approach to incorporate error checking into the design process. In *ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 273–284. American Society of Mechanical Engineers, 2008.

[12] Tolga Kurtoglu, Irem Y Tumer, and David C Jensen. A Functional Failure Reasoning Methodology for Evaluation of Conceptual System Architectures. *Research in Engineering Design*, 21(4):209–234, 2010.

[13] Ryan S Hutcheson, Daniel A McAdams, Robert B Stone, and Irem Y Tumer. A Function-Based Methodology for Analyzing Critical Events. In *Proceedings of IDETC/CIE*, 2006.

[14] Clifton A Ericson and Clifton Ll. Fault tree analysis. *Hazard analysis techniques for system safety*, pages 183–221, 2000.

[15] Clifton A Ericson. Event tree analysis. *Hazard Analysis Techniques for System Safety*, pages 223–234.

[16] Venkat Venkatasubramanian, Jinsong Zhao, and Shankar Viswanathan. Intelligent systems for hazop analysis of complex process plants. *Computers & Chemical Engineering*, 24(9):2291–2302, 2000.

[17] Erik Hollnagel. *Barriers and accident prevention.* Ashgate Publishing, Ltd., 2004.

[18] Chris W Johnson and C Michael Holloway. The esa/nasa soho mission interruption: Using the stamp accident analysis technique for a software related ŚmishapŠ. *Software: Practice and Experience*, 33(12):1177–1198, 2003.

Chapter 3

[1] Paul R Wiese and Philip John. *Engineering Design in the Multi-discipline Era: A Systems Approach.* Wiley, 2003.

[2] Enrico Zio. Reliability engineering: Old problems and new challenges. *Reliability Engineering & System Safety*, 94(2):125–141, 2009.

[3] Nancy Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety.* MIT Press, 2011.

[4] Erik Hollnagel, David D Woods, and Nancy Leveson. *Resilience Engineering: Concepts and Precepts.* Ashgate Publishing, Ltd., 2007.

[5] EC Baroth, June Zakrajsek, WT Powers, J Fox, B Prosser, J Pallix, and K Schweikard. Ivhm (integrated vehicle health management) techniques for future space vehicles. In *37th Joint Propulsion Conference*, 2001.

[6] Nancy G Leveson. The need for new paradigms in safety engineering. In *Safety-Critical Systems: Problems, Process and Practice*, pages 3–20. Springer, 2009.

[7] IEC. 61508 functional safety of electrical/electronic/programmable electronic safety-related systems. *International electrotechnical commission*, 1998.

[8] SAE ARP4761. Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *SAE International, December*, 1996.

[9] SAE ARP4754. Certification considerations for highly-integrated or complex aircraft systems. *Society of Automotive Engineers Inc*, 1996.

[10] Benjamin S Blanchard. *System Engineering Management*, volume 64. Wiley. com, 2012.

[11] Dennis M Buede. *The Engineering Design of Systems: Models and Methods*, volume 55. John Wiley & Sons, 2011.

[12] Julie Hirtz, Robert B Stone, Daniel A McAdams, Simon Szykman, and Kristin L Wood. A Functional Basis For Engineering Design: Reconciling And Evolving Previous Efforts. *Research in engineering Design*, 13(2):65–82, 2002.

[13] Robert L Nagel, Robert B Stone, Ryan S Hutcheson, Daniel A McAdams, and Joseph A Donndelinger. Function Design Framework (FDF): Integrated Process And Function Modeling For Complex Systems. In *ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2008)*, pages 273–286, 2008.

[14] Tolga Kurtoglu and Matthew I Campbell. Automated Synthesis Of Electromechanical Design Configurations From Empirical Analysis Of Function To Form Mapping. *Journal of Engineering Design*, 20(1):83–104, 2009.

[15] IEEE1220. *IEEE Standard for Application and Management of the Systems Engineering Process.* IEEE New York, NY, USA., 2005.

[16] ISO-IEC15288. *Systems Engineering Ű System Life Cycle Processes.* International Standardization Organization., 2002.

[17] Mary Ann Lundteigen, Marvin Rausand, and Ingrid Bouwer Utne. Integrating rams engineering and management with the safety life cycle of iec 61508. *Reliability Engineering & System Safety*, 94(12):1894–1903, 2009.

[18] OMG OMG. Unified modeling language (omg uml), 2007.

[19] IBM. Rational doors., 2010.

[20] GeenSys. Reqtify, 2008.

[21] T. A. Henzinger, P. Ho, and H Wong-Toi. Hytech: A Model Checker for Hybrid Systems. *Electronics Research Laboratory, College of Engineering, University of California.*, 1997.

[22] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-time Systems. *Information and Computation*, 111(2):193–244, 1994.

[23] Israel Santiago Barragan, Matthias Roth, Jean-Marc Faure, et al. Obtaining temporal and timed properties of logic controllers from fault tree analysis. In

Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006, Saint-Etienne, France, 2006.

[24] Dominique Evrot, Jean-Francois Petin, and Dominique Mery. Formal specification of safe manufacturing machines using the b method: Application to a mechanical. In *Information Control Problems in Manufacturing*, volume 12, pages 281–286, 2006.

[25] Irem Y Tumer, Robert B Stone, and David G Bell. Requirements for a failure mode taxonomy for use in conceptual design. In *Proceedings of the International Conference on Engineering Design, ICED*, volume 3. Paper 1612, Stockholm,, Sweden, 2003.

[26] Robert B Stone, Irem Y Tumer, and Michael E Stock. Linking product functionality to historic failures to improve failure analysis in design. *Research in Engineering Design*, 16(1-2):96–108, 2005.

[27] Tolga Kurtoglu and Irem Y Tumer. A graph-based fault identification and propagation framework for functional design of complex systems. *Journal of Mechanical Design*, 130(5):051401, 2008.

[28] Irem Y Tumer and Carol S Smidts. Integrated design-stage failure analysis of software-driven hardware systems. *Computers, IEEE Transactions on*, 60(8):1072–1084, 2011.

[29] Edward Balaban, Abhinav Saxena, Kai Goebel, C Byington, Matthew Watson, Sudarshan Bharadwaj, Matthew Smith, and S Amin. Experimental Data Collec-

tion And Modeling For Nominal And Fault Conditions On Electro-mechanical Actuators. In *Annual Conference of the Prognostics and Health Management Society*, pages 1–15, 2009.

[30] Jean-Christophe Blaise, Pascal Lhoste, and Joseph Ciccotelli. Formalisation of normative knowledge for safe design. *Safety Science*, 41(2):241–261, 2003.

[31] AM Madni. Designing for resilience. *ISTI Lecture Notes on Advanced Topics in Systems Engineering*, 2007.

[32] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning Assumptions For Compositional Verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.

[33] Dimitra Giannakopoulou, Corina S Păsăreanu, and Howard Barringer. Component Verification With Automatically Generated Assumptions. *Automated Software Engineering*, 12(3):297–320, 2005.

[34] Wonhong Nam and Rajeev Alur. Learning-based Symbolic Assume-guarantee Reasoning With Automatic decomposition. In *Automated Technology for Verification and Analysis*, pages 170–185. Springer, 2006.

[35] S. Chaki, E. Clarke, N. Sinha, and P. Thati. Automated Assume-guarantee Reasoning for Simulation Conformance. In *Computer Aided Verification*, pages 241–246. Springer, 2005.

[36] Roberto WS Rodrigues. Formalising UML Activity Diagrams Using Finite State Processes. In *Proc. of the 3rd Intl. Conf. on the Unified Modeling Language, York, UK*. Citeseer, 2000.

[37] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *Computer Aided Verification*, pages 440–451. Springer, 1998.

[38] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In *Proceedings of the 26th International Conference on Software Engineering*, pages 211–220. IEEE Computer Society, 2004.

Chapter 4

[1] B. Saporito. How Safe is the Boeing 787? See also URL http://business.time.com/2013/01/11/how-safe-is-the-boeing-787/, Jan. 2013.

[2] Hoda A ElMaraghy. Changing and evolving products and systems–models and enablers. In *Changeable and reconfigurable manufacturing systems*, pages 25–45. Springer, 2009.

[3] American Institute of Aeronautics and Astronautics Staf. *AIAA Guide for the Verification and Validation of Computational Fluid Dynamics Simulations.* American Institute of Aeronautics & Astronautics, 1998.

[4] Robert Plant and Rose Gamble. Methodologies for the development of knowledge-based systems, 1982-2002. *The Knowledge Engineering Review*, 18(1):47–81, 2003.

[5] Harry Foster. *Prologue: The 2012 Wilson Research Group Functional Verification Study,*. American Institute of Aeronautics & Astronautics, 2013.

[6] Collett International Research Inc. *2002 IC/ASIC Functional Verification Study*. 2002.

[7] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.

[8] Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[9] Joseph T Buck, Soonhoi Ha, Edward A Lee, and David G Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. 1994.

[10] Matlab Simulink and MA Natick. The mathworks. *Inc., Natick, MA*, 1993.

[11] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Elsevier, 2011.

[12] T. Henzinger and J. Sifakis. The Embedded Systems Design Challenge. 4085:1–15.

[13] A. Pnueli. The Temporal Logic of Programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[14] Dimitra Giannakopoulou, Corina S Pasareanu, and Howard Barringer. Assumption Generation for Software Component Verification. In *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, pages 3–12. IEEE, 2002.

[15] Jamieson Cobleigh, Dimitra Giannakopoulou, and Corina Păsăreanu. Learning Assumptions for Compositional Verification. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346, 2003.

[16] Sagar Chaki, Edmund Clarke, Nishant Sinha, and Prasanna Thati. Automated Assume-guarantee Reasoning for Simulation Conformance. In *Computer Aided Verification*, pages 241–246. Springer, 2005.

[17] Rajeev Alur, P Madhusudan, and Wonhong Nam. Symbolic Compositional Verification by Learning Assumptions. In *Computer Aided Verification*, pages 289–292. Springer, 2005.

[18] Shing Chi Cheung and Jeff Kramer. Checking Safety Properties Using Compositional Reachability Analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(1):49–78, 1999.

[19] Tolga Kurtoglu, M Campbell, C Bryant, R Stone, and D McAdams. Deriving a component basis for computational functional synthesis. In *International conference on engineering design, ICED*, volume 5, pages 15–18, 2005.

[20] Tolga Kurtoglu and Irem Y Tumer. Ffip: A framework for early assessment of functional failures in complex systems. In *International Conference on Engineering Design*, 2007.

[21] Enoch Y Wang and Betty HC Cheng. Formalizing the functional model within object-oriented design. *International Journal of Software Engineering and Knowledge Engineering*, 10(01):5–30, 2000.

[22] Julie Hirtz, Robert B Stone, Daniel A McAdams, Simon Szykman, and Kristin L Wood. A functional basis for engineering design: reconciling and evolving previous efforts. *Research in engineering Design*, 13(2):65–82, 2002.

[23] Robert B Stone and Kristin L Wood. Development of a functional basis for design. *Journal of Mechanical Design*, 122:359, 2000.

[24] Roberto WS Rodrigues. Formalising UML Activity Diagrams Using Finite State Processes. In *Proc. of the 3rd Intl. Conf. on the Unified Modeling Language, York, UK*. Citeseer, 2000.

[25] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Ltsa-ws: a tool for model-based verification of web service compositions and choreography. In *Proceedings of the 28th international conference on Software engineering*, pages 771–774. ACM, 2006.

[26] William F Gilreath. Concurrency state models and java programs. *Scalable Computing: Practice and Experience*, 3(4), 2001.

[27] H. Garavel, M. Sighireanu, et al. A Graphical Parallel Composition Operator for Process Algebras. In *International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV*, volume 99, pages 185–202. Citeseer, 1999.

[28] Scott Poll, Ann Patterson-Hine, Joe Camisa, David Garcia, David Hall, Charles Lee, Ole J Mengshoel, Christian Neukom, David Nishikawa, John Ossenfort, et al. Advanced diagnostics and prognostics testbed. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 178–185, 2007.

[29] Michael Tiller. *Introduction to physical modeling with Modelica*, volume 615. Kluwer Academic Pub, 2001.

[30] H. Mehrpouyan, D. C. Jensen, C. Hoyle, I. Y. Tumer, and T. Kurtgolu. A Model-based Failure Identification and Propagation Framework for Conceptual Design of Complex Systems. *Proceedings of the ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2012.

[31] Jeff Magee, Jeff Kramer, and Dimitra Giannakopoulou. Behaviour analysis of software architectures. *Software Architecture*, 12:35, 1999.

[32] Charles Jason Woodard. *Architectural strategy and design evolution in complex engineered systems.* PhD thesis, Citeseer, 2006.

Chapter 5

[1] N. Leveson, N. Dulac, D. Zipkin, J. Cutcher-Gershenfeld, J. Carroll, and B. Barrett. Engineering Resilience into Safety-critical Systems. *Resilience Engineering–Concepts and Precepts. Ashgate Aldershot*, pages 95–123, 2006.

[2] E. Jen. *Robust Design: A Repertoire of Biological, Ecological, and Engineering Case Studies*. Oxford University Press, USA, 2005.

[3] A.M. Madni and S. Jackson. Towards a Conceptual Framework for Resilience Engineering. *Systems Journal, IEEE*, 3(2):181–191, 2009.

[4] Stephen W Ormon, C Richard Cassady, and Allen G Greenwood. Reliability Prediction Models to Support Conceptual Design. *Reliability, IEEE Transactions on*, 51(2):151–157, 2002.

[5] Ying Zhang, Tolga Kurtoglu, Irem Y Tumer, and Bryan O'Halloran. System–Level Reliability Analysis for Conceptual Design of Electrical Power Systems. In *Conference on Systems Engineering Research (CSER)*, pages 15–16, 2011.

[6] JK Gershenson, GJ Prasad, and Y. Zhang. Product Modularity: Definitions and Benefits. *Journal of Engineering design*, 14(3):295–313, 2003.

[7] Byeng D. Youn, Chao Hu, and Pingfeng Wang. Resilience-driven system design of complex engineered systems. *Journal of Mechanical Design*, 133(10):101011–15, 2011.

[8] Hoda Mehrpouyan, Brandon Haley, Andy Dong, Irem Y Tumer, and Chris Hoyle. Resilient design of complex engineered systems. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V03AT03A048–V03AT03A048. American Society of Mechanical Engineers, 2013.

[9] R.B. Stone, I.Y. Tumer, and M. Van Wie. The Function-failure Design Method. *Journal of Mechanical Design*, 127(3):397–407, 2005.

[10] Tolga Kurtoglu, Irem Y Tumer, and David C Jensen. A Functional Failure Reasoning Methodology for Evaluation of Conceptual System Architectures. *Research in Engineering Design*, 21(4):209–234, 2010.

[11] Nestor F Michelena and Panos Y Papalambros. A hypergraph framework for optimal model-based decomposition of design problems. *Computational Optimization and Applications*, 8(2):173–196, 1997.

[12] C McCulley and CL Bloebaum. A genetic tool for optimal design sequencing in complex engineering systems. *Structural Optimization*, 12(2-3):186–201, 1996.

[13] J. Ash and D. Newth. Optimizing Complex Networks for Resilience Against Cascading Failure. *Physica A: Statistical Mechanics and its Applications*, 380:673–683, 2007.

[14] James P Bagrow, Sune Lehmann, and Yong-Yeol Ahn. Robustness and modular structure in networks. *arXiv preprint arXiv:1102.5085*, 2011.

[15] Katja Holtta, Eun Suk Suh, and Olivier de Weck. Tradeoff between modularity and performance for engineered systems and products. In *ICED 05: 15th International Conference on Engineering Design: Engineering Design and the Global Economy*, page 2820. Engineers Australia, 2005.

[16] Kosuke Ishii and Tae G Yang. Modularity: international industry benchmarking and research roadmap. In *ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 1–11. American Society of Mechanical Engineers, 2003.

[17] Dean H Stamatis. *Failure Mode Effect Analysis: FMEA from Theory to Execution.* ASQ Quality Press, 2003.

[18] H. Jeong R. Albert and A.-L. Barabási. Error and Attack Tolerance of Complex Networks. *Nature*, 6794(406):378ï£¡–382, July 2007.

[19] P. Crucitti, V. Latora, M.Marchiori, and A. Rapisarda. Error and attack tolerance of complex networks. *Physica A: Statistical Mechanics and its Applications*, 1–3(340):388ï£¡–394, 2004.

[20] K. I. Goh, E. Oh, H. Jeong, B. Kahng, and D. Kim. Classification of scale-free networks. *Proceedings National Academy of Sciences,*, 20(99):12583ï£¡12588, October 2002.

[21] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A. L. Barabï£¡si. The Large-scale Organization of Metabolic Networks. *Nature*, 6804(407):651–654, 2000.

[22] Y. Tu. How Robust Is The Internet? *Nature*, 406:353–354, 2000.

[23] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin. Resilience of the Internet to Random breakdowns. *Phys. Rev. Lett*, 85(407):4626ï£¡4628, 2000.

[24] Duncan S Callaway, Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Network robustness and fragility: Percolation on random graphs. *Physical review letters*, 85(25):5468, 2000.

[25] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 25–34. IEEE, 2003.

[26] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. Epidemic Thresholds in Real Networks. *ACM Transactions on Information and System Security (TISSEC)*, 10(4):1, 2008.

[27] Yamir Moreno, Romualdo Pastor-Satorras, and Alessandro Vespignani. Epidemic Outbreaks in Complex Heterogeneous Networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 26(4):521–529, 2002.

[28] Kristina Lerman and Rumi Ghosh. Information Contagion: An Empirical Study of the Spread of News on Digg and Twitter Social Networks. In *Proceedings of 4th International Conference on Weblogs and Social Media (ICWSM)*, 2010.

[29] Daron Acemoglu, Asuman Ozdaglar, and Ali ParandehGheibi. Spread of (mis) Information in Social Networks. *Games and Economic Behavior*, 70(2):194–227, 2010.

[30] M. Tiller. *Introduction to Physical Modeling with Modelica*, volume 615. Springer, 2001.

[31] J.A. Fax and R.M. Murray. Information Flow and Cooperative Control of Vehicle Formations. *Automatic Control, IEEE Transactions on*, 49(9):1465–1476, 2004.

[32] A. Jamakovic and S. Uhlig. On the Relationship Between the Algebraic Connectivity and Graph's Robustness to Node and Link Failures. In *Next Generation Internet Networks, 3rd EuroNGI Conference on*, pages 96–102. IEEE, 2007.

[33] J. Wu, M. Barahona, Y.J. Tan, and H.Z. Deng. Spectral Measure of Structural Robustness in Complex Networks. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(6):1244–1252, 2011.

[34] A. Jamakovic, RE Kooij, P. Van Mieghem, and E.R. van Dam. Robustness of Networks Against Viruses: the Role of the Spectral Radius. In *Communications and Vehicular Technology, 2006 Symposium on*, pages 35–38. IEEE, 2006.

[35] S. Sarkar and A. Dong. Community Detection in Graphs Using Singular Value Decomposition. *Physical Review E*, 83(4):046114, 2011.

[36] S. Sarkar, A. Dong, J. A. Henderson, and P. A. Robinson. Spectral Fingerprints of Modularity and Hierarchy in Networks. *PLOS One*, 2012.

[37] Theodore E Harris. Contact interactions on a lattice. *The Annals of Probability*, pages 969–988, 1974.

[38] Joaquín Marro and Ronald Dickman. *Nonequilibrium Phase Transitions in Lattice Models*. Cambridge University Press, 2005.

[39] Rick Durrett. Stochastic Spatial Models. *Siam Review*, 41(4):677–718, 1999.

[40] Deepayan Chakrabarti, Jure Leskovec, Christos Faloutsos, Samuel Madden, Carlos Guestrin, and Michalis Faloutsos. Information survival threshold in sensor and p2p networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1316–1324. IEEE, 2007.

[41] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O.J. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, et al. Advanced Diagnostics and Prognostics Testbed. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 178–185, 2007.

Chapter 6

[1] US Military Standard. Mil-std-1629a (1980). *Procedures for performing a failure mode, effect and criticality analysis. Department of Defense, USA*.

[2] Norman Hailstone Roberts. *Fault Tree Handbook*. Systems and Reliability Research, Office of Nuclear Regulatory Research, US Nuclear Regulatory Commission, 1981.

[3] Gerhard Pahl, Ken Wallace, and Lucièenne Blessing. *Engineering Design: A Systematic Approach*, volume 157. Springer, 2007.

[4] Karl T Ulrich. *Product Design And Development*. McGraw-Hill Education, 2003.

[5] Kevin N Otto and Kristin L Wood. *Product Design: Techniques In reverse Engineering And New Product Development*. Prentice-Hall New York, 2001.

[6] Benjamin S Blanchard, Wolter J Fabrycky, and Wolter J Fabrycky. *Systems Engineering And Analysis*, volume 5. Prentice Hall International Series in Industrial and Systems Engineering, 2010.

[7] Defense Acquisition Guidebook DAG and Workplace Ethic. Introduction To Systems Engineering. 2000.

[8] Marco Bozzano and Adolfo Villafiorita. Improving System Reliability Via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform. In *Computer Safety, Reliability, and Security*, pages 49–62. Springer, 2003.

[9] Julie Hirtz, Robert B Stone, Daniel A McAdams, Simon Szykman, and Kristin L Wood. A Functional Basis For Engineering Design: Reconciling And Evolving Previous Efforts. *Research in engineering Design*, 13(2):65–82, 2002.

[10] Robert L Nagel, Robert B Stone, Ryan S Hutcheson, Daniel A McAdams, and Joseph A Donndelinger. Function Design Framework (FDF): Integrated Process And Function Modeling For Complex Systems. In *ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2008)*, pages 273–286, 2008.

[11] Robert B Stone and Kristin L Wood. Development Of A Functional Basis For Design. *Journal of Mechanical Design*, 122:359, 2000.

[12] Tolga Kurtoglu and Matthew I Campbell. Automated Synthesis Of Electromechanical Design Configurations From Empirical Analysis Of Function To Form Mapping. *Journal of Engineering Design*, 20(1):83–104, 2009.

[13] Sambasiva Bhatta, Ashok Goel, and Sattiraju Prabhakar. Innovation In Analogical Design: A Model-based Approach. In *Artificial Intelligence in DesignŠ94*, pages 57–74. Springer, 1994.

[14] Francois E Cellier, Hilding Elmqvist, and Martin Otter. Modeling From Physical Principles. In *The Control Handbook*. Citeseer, 1996.

[15] Peter Fritzson. *Principles Of Object-oriented Modeling And Simulation With Modelica 2.1*. Wiley. com, 2010.

[16] MATLAB UserŠs Guide. The Mathworks. *Inc., Natick, MA*, 5, 1998.

[17] Tolga Kurtoglu and Irem Y Tumer. A Graph-based Fault Identification And Propagation Framework For Functional Design Of Complex Systems. *Journal of Mechanical Design*, 130:051401, 2008.

[18] Robert B Stone, Irem Y Tumer, and Michael E Stock. Linking Product Functionality To Historic Failures To Improve Failure Analysis In Design. *Research in Engineering Design*, 16(1-2):96–108, 2005.

[19] Robert B Stone, Irem Y Tumer, and Michael Van Wie. The Function-failure Design Method. *Journal of Mechanical Design*, 127:397, 2005.

[20] Katie Grantham Lough, Robert Stone, and Irem Y Tumer. The Risk In Early Design Method. *Journal of Engineering Design*, 20(2):155–173, 2009.

[21] Katie Grantham Lough, Robert B Stone, and IY Tumer. Implementation Procedures For The Risk In Early Design (RED) Method. *J Ind Syst Eng*, 2(2):126–143, 2008.

[22] Daniel Krus and Katie Grantham Lough. Applying Function-based Failure Propagation In Conceptual Design. ASME, 2007.

[23] Zhaofeng Huang and Yan Jin. Extension of Stress and Strength Interference Theory for Conceptual Design-for-Reliability. *Journal of Mechanical Design*, 131(7):71001, 2009.

[24] Sergei N Dorogovtsev and José FF Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. OUP Oxford, 2003.

[25] Duncan J Watts. *Small Worlds: the Dynamics of Networks Between Order and Randomness*. Princeton university press, 2003.

[26] Hongquan Jiang, Jianmin Gao, and Fumin Chen. System Failure Analysis Based on Complex Network Theory. In *Reliability and Maintainability Symposium, 2009. RAMS 2009. Annual*, pages 176–181. IEEE, 2009.

[27] Sarah A Sheard and Ali Mostashari. A Complexity Typology for Systems Engineering. *Syst. Eng*, 2009.

[28] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 25–34. IEEE, 2003.

[29] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. Epidemic Thresholds in Real Networks. *ACM Transactions on Information and System Security (TISSEC)*, 10(4):1, 2008.

[30] Yamir Moreno, Romualdo Pastor-Satorras, and Alessandro Vespignani. Epidemic Outbreaks in Complex Heterogeneous Networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 26(4):521–529, 2002.

[31] F. Boschetti, M. Prokopenko, I. Macreadie, and A.M. Grisogono. Defining and Detecting Emergence in Complex Networks. In *Knowledge-based intelligent information and engineering systems*, pages 905–905. Springer, 2005.

[32] Elaine Aspinwall. Models In Design For Reliability Optimisation. 2013.

[33] Way Kuo and V Rajendra Prasad. An Annotated Overview Of System-Reliability Optimization. *Reliability, IEEE Transactions on*, 49(2):176–187, 2000.

[34] Ranjan Kumar, Kazuhiro Izui, Masataka Yoshimura, and Shinji Nishiwaki. Multi-objective Hierarchical Genetic Algorithms For Multilevel Redundancy Allocation Optimization. *Reliability Engineering & System Safety*, 94(4):891–904, 2009.

[35] Krishna B Misra. *Reliability Analysis And Prediction: A Methodology Oriented Treatment*. Access Online via Elsevier, 1992.

[36] AO Charles Elegbede, Chengbin Chu, Kondo H Adjallah, and Farouk Yalaoui. Reliability Allocation Through Cost Minimization. *Reliability, IEEE Transactions on*, 52(1):106–111, 2003.

[37] Way Kuo. *Optimal Reliability Design: Fundamentals And Applications*. Cambridge university press, 2001.

[38] Rajeev Alur, P Madhusudan, and Wonhong Nam. Symbolic Compositional Verification By Learning Assumptions. In *Computer Aided Verification*, pages 548–562. Springer, 2005.

[39] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning Assumptions For Compositional Verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.

[40] Dimitra Giannakopoulou, Corina S Păsăreanu, and Howard Barringer. Component Verification With Automatically Generated Assumptions. *Automated Software Engineering*, 12(3):297–320, 2005.

[41] Wonhong Nam and Rajeev Alur. Learning-based Symbolic Assume-guarantee Reasoning With Automatic decomposition. In *Automated Technology for Verification and Analysis*, pages 170–185. Springer, 2006.

[42] Sagar Chaki, Edmund Clarke, Nishant Sinha, and Prasanna Thati. Automated Assume-guarantee Reasoning For Simulation Conformance. In *Computer Aided Verification*, pages 534–547. Springer, 2005.

[43] Edward Balaban, Abhinav Saxena, Kai Goebel, C Byington, Matthew Watson, Sudarshan Bharadwaj, Matthew Smith, and S Amin. Experimental Data Collection And Modeling For Nominal And Fault Conditions On Electro-mechanical Actuators. In *Annual Conference of the Prognostics and Health Management Society*, pages 1–15, 2009.

[44] Roberto WS Rodrigues. Formalising UML Activity Diagrams Using Finite State Processes. In *Proc. of the 3rd Intl. Conf. on the Unified Modeling Language, York, UK*. Citeseer, 2000.