Towards A Framework For Resilient Design Of Complex Engineered Systems

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. The proposed approach is validated on the quad-redundant Electro-Mechanical Actuator (EMA) of a Flight Control Surface (FCS) of an aircraft.

1 Introduction

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 [1, 2]. 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. 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 [3].

In order to design a resilient system, first we define the concept of complex systems. System of systems (SOS) is a collection of other elements which themselves are distinct complex systems that interact with one another to achieve a common goal. The more 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 [4], Bedau [5], Chalmers [6], and Seager [7] categorized these emergent properties into three distinct types: (1) normal emergence, (2) weak emergence, and (3) strong emergence.

The three categories of emergence are depicted in Figure 1. According to Bedau and Chalmers [5, 6], 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. [8] looks at an air traffic control system as a good example of normal emergence. In this system, a specific and fixed role is assigned to Flight Management System (FMS), Instrument Landing System (ILS), and the air traffic controller. The function of a designed system with normal emergence is an intended emergent property of the planned interaction of the individual parts and components.

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 [9]. 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 [10] 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

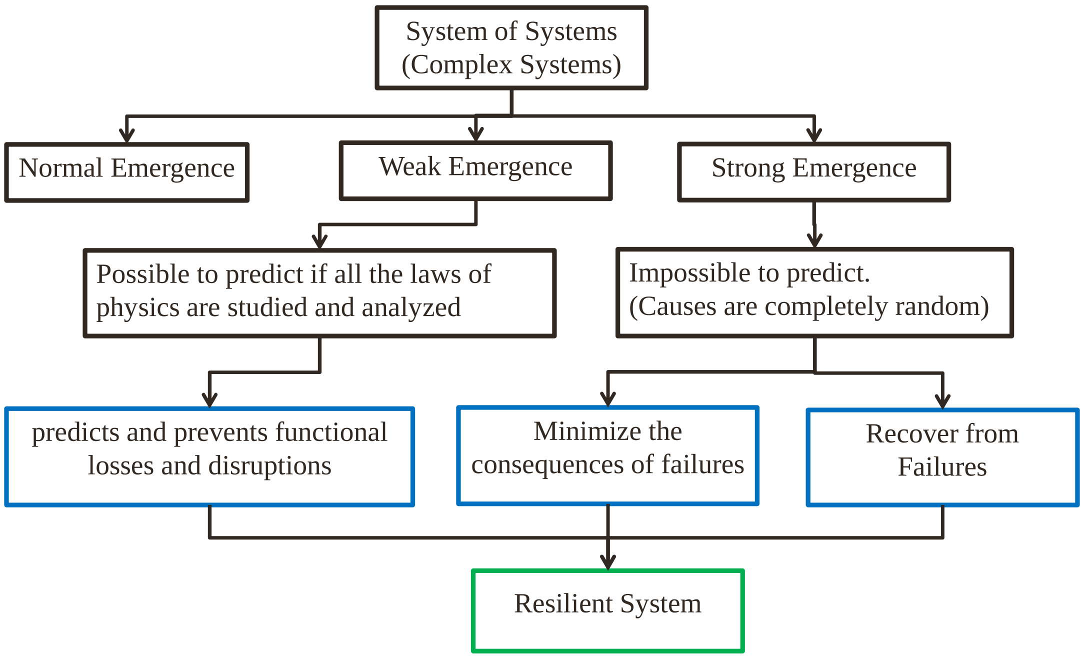

System of Systems (Complex Systems)
Normal Emergence
Weak Emergence
Strong Emergence
Possible to predict if all the laws of physics are studied and analyzed
Impossible to predict. (Causes are completely random)
predicts and prevents functional losses and disruptions
Minimize the consequences of failures
Recover from Failures
Resilient System

Fig. 1: Resilience and Emergence in Systems.

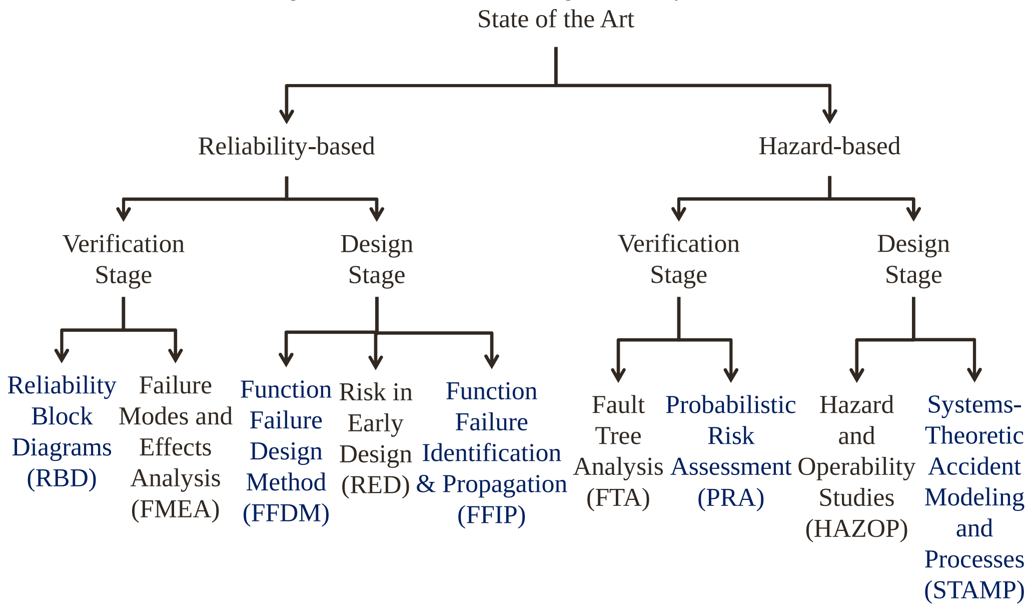

State of the Art
Reliability-based
Hazard-based
Verification Stage
Design Stage
Verification Stage
Design Stage
Reliability Block Diagrams (RBD)
Failure Modes and Effects Analysis (FMEA)
Function Failure Design Method (FFDM)
Risk in Early Design (RED)
Function Failure Identification & Propagation (FFIP)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

Fig. 2: Techniques for Safety and Reliability Analysis of System Design.

entire system, including aircraft and pilot could have been more resilient and adaptable to a range of external and internal threats and failures.

In this paper, resilience is viewed as a system's ability to cope with complexity [11] and adapt to changes caused by emergence, either weak or strong. As depicted in Figure 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

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.

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. In section *3* an overview of the step-by-step implementation of the framework, both at the component and system level, is explained. 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.

2 Background

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, two distinct categories of methods are usually adopted to address safety analysis of system design. First, reliability-based approaches are based on identifying fault 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 that path. Hence, hazard-based techniques are system state centric whereas the reliability-based approaches are fault centric.

2.1 Reliability-based Techniques

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.

2.1.1 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) [12], 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) [13] 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 is 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 Stone et al. [14,15] 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. Risk in Early Design (RED) method presented by Grantham et al. [16, 17] is built upon the FFDM technique which formulates the functional-failure likeliness and consequence associated with each function failure. Nevertheless, Devendorf [18] 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 syntax.

In order to overcome these limitations, other research efforts have drawn attention to the importance of failure cascades in reliability analysis. Kurtoglu at al. [19] 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.

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) [20] 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 a significant amount of resources. Another form of symbolic logic modeling technique is known as Event Tree Analysis (ETA) [21] 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.

Probabilistic Risk Assessment (PRA) [22] is a technique used for analysis of failure risk [23]. PRA incorporates a number of fault/event modeling methods, such as event sequence diagrams and fault trees, and integrates them into a probabilistic analysis to guide decision-making during verification stage. However, the requirement for developing a fully specified system model as part of the verification process is constable. The reason is that such detailed, high-fidelity models of complex systems are not available during early stages.

2.2.2 Design Stage Approaches

Another technique for safety analysis is Hazard and Operability Studies (HAZOP) [24] that is based on modeling the interaction flow between components and recognizing a hazard if components deviate from the intended operation of designs. 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.

In systematic models, such as Systems-Theoretic Accident Modeling and Processes (STAMP), accidents result from several causal factors that occur unexpectedly in a specific time and space [25]. 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.

There are many benefits in using STAMP models as the basis for hazard analysis of a complex system. However, Johnson et al. [26] 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, [26] 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 conclusions regarding the cause of failure in the system under study.

3 Methodology

The techniques reviewed in the previous section 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), proving that the system under consideration satisfies its safety requirements.

The proposed framework relies on constructing a finite model of a design and checking it against its desired safety properties. In the context of this work, the properties of a system are modeled as transition systems. A desired *safety* property 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. Another advantage of modeling the system as 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 [27] 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 guarantee 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* [28–32] 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 used as a tool to convert each design into an adjacency matrix of nodes (components) and edge connections [33]. Subsequently, Non-Linear Dynamical System (NLDS) and epidemic spreading algorithms are used to analyze the propagation of failure in complex engineered system design.

Furthermore, the framework 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.

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 a system *has*. Therefore, the proof of safety is only conveyed by the absence of failures and accidents. For this reason, safety-proofing a system design is never absolute or complete. However, in this research,

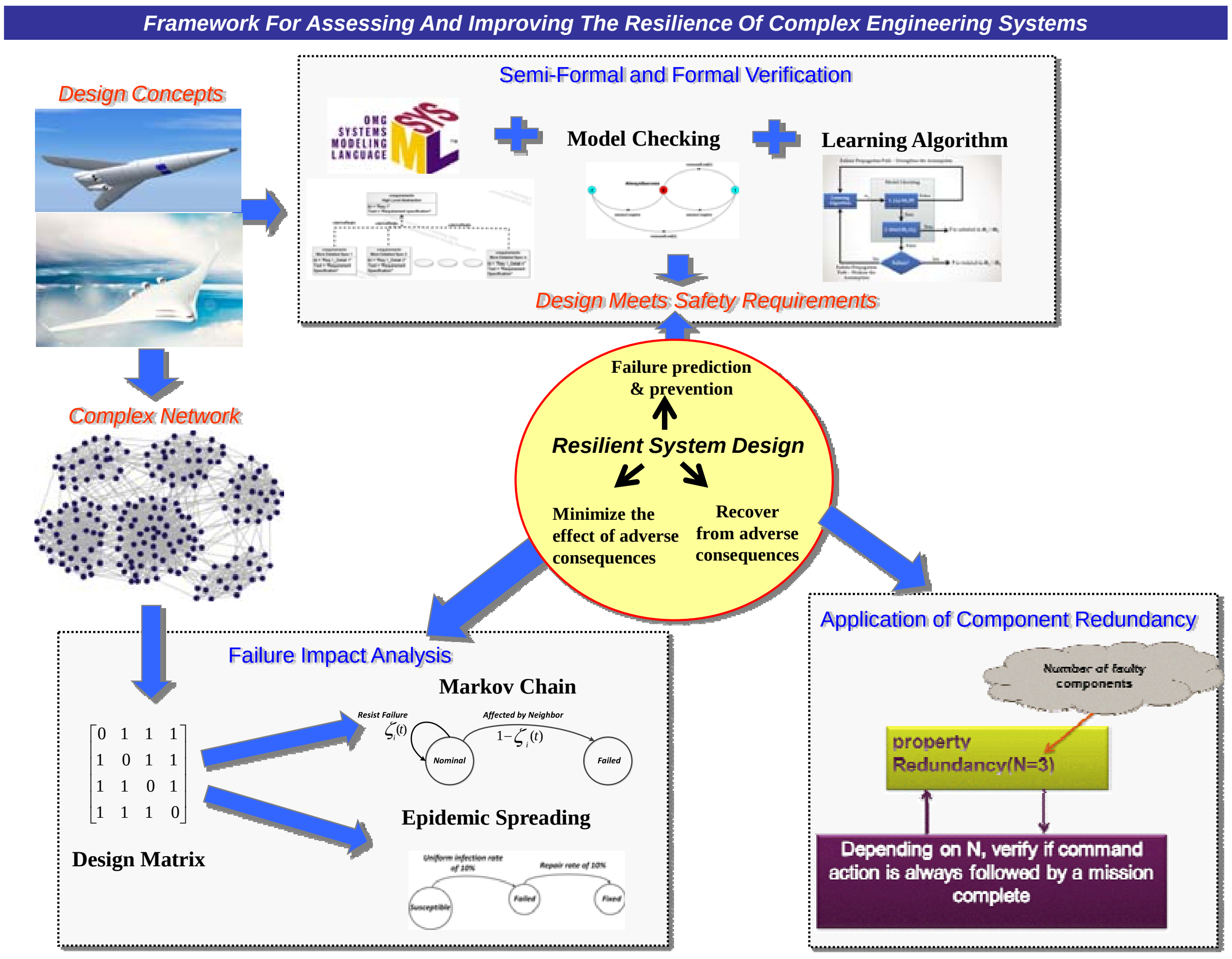

Framework For Assessing And Improving The Resilience Of Complex Engineering Systems
Design Concepts
Semi-Formal and Formal Verification
Model Checking
Learning Algorithm
Design Meets Safety Requirements
Complex Network
Failure prediction & prevention
Resilient System Design
Minimize the effect of adverse consequences
Recover from adverse consequences
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
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

Fig. 3: An Overview of The Proposed Approach.

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. 3 represents an overview of the approach which will be discussed in detail in the next section.

3.1 Formal and Semi-Formal Verification Approaches

Design requirements are the specification of safety constraints initially defined in the design [34]. 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,

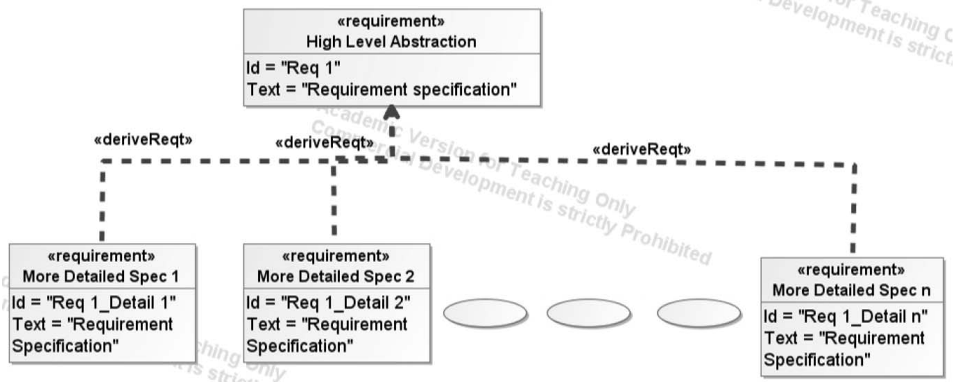


Fig. 4: Requirements Decomposition.

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.

3.1.1 Safety Requirements Modeling Using SysML

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) [35] to support various aspect of system modeling, Rational Doors [36] to express the requirements, and Reqtify [37] 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 engineers 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 [38] profile was developed borrowing a subset of the UML language to meet the requirements of a general purposed language for system engineering.

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. 4 depicts the decomposition of a single abstract requirement into several more explicit ones. A study by Blaise et al. [39] 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. 5 illustrates how a single requirement can be satisfied by a set of sub-systems 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 sub-systems 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. [40] demonstrated, safety is a changing characteristic of complex systems that, once integrated into the design, is not preserved unless enforced throughout system operation. 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. 6 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.

After decomposing 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.

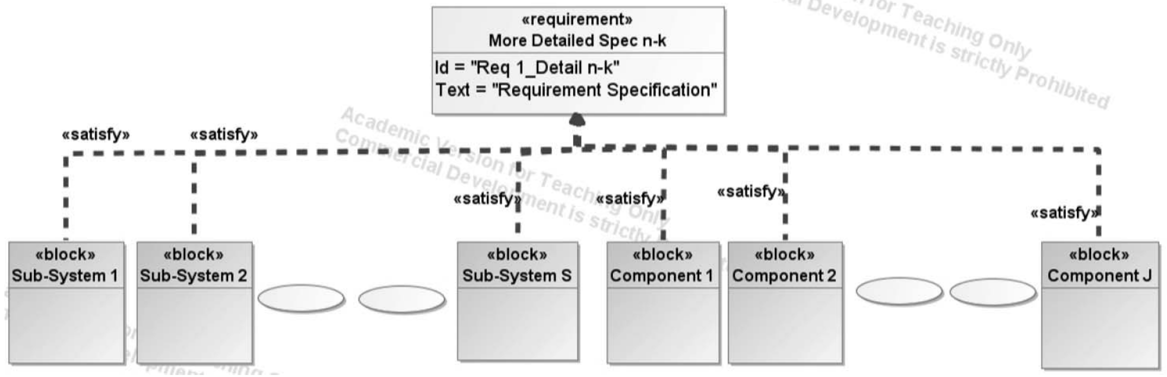


Fig. 5: Requirements Mapping.

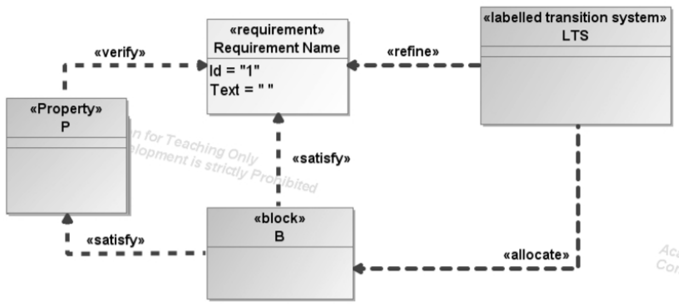


Fig. 6: Requirements Traceability.

3.1.2 Automated Assume-guarantee Reasoning

In [41], 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, system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time [42]. 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) [43, 44]. 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) [45] technique. CRA incrementally composes and abstracts the component models into subsystem and, ultimately, a high-level system models.

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 [32, 46]:

Rule ASYM

$$\frac{\begin{array}{l} 1 : \langle A \rangle \ M_1 \ \langle P \rangle \\ 2 : \langle true \rangle \ M_2 \ \langle A \rangle \end{array}}{\langle true \rangle \ M_1 \parallel M_2 \ \langle P \rangle}$$

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 ($\langle true \rangle \ M_1 \parallel M_2 \ \langle P \rangle$).

In this research, the algorithm in [47] is used to automatically generate assume-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 automated design verification proves 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.

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. 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 $s_0 \in S$

This type of graphical modeling, however, could easily become unmanageable for large complex systems. Therefore, an algebraic notation known as Finite State Process (FSP) [48] is used to define the behavior of processes in a design. FSP is a specification language as opposed to modeling language with semantics defined in terms of LTSs. Every FSP model has a corresponding LTS description and vice versa.

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. Then the verification algorithm analyzes all execution paths of the composed model to ensure the specified property holds for all executions of the system. As a result, some of the weak emergence can be predicted and prevented because most of the laws of physics are taken into the consideration and exhaustive simulation and verification is conducted.

On the other hand, strong emergence is one of the most difficult type of emergence to be identified. By nature, this type of emergence results from completely random factors. Therefore, it is essential for complex systems to be designed in a way that are able to survive and recover from unexpected disruptions and operational environment degradations [49]. This differs from traditional definitions of reliability which only deals with the functional response of components and (sub)systems. In this research reliability analysis is part of the design process 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) [50], the Functional Failure Identification and Propagation (FFIP) [19], and decomposition-based design optimization [51, 52] 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.

3.2 Design Topology and its Effect on Failure Propagation

In the past several years, scientific interest has been devoted to modeling and characterization of complex systems that are defined as networks [53, 54]. 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.

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 [55], and has wide applications in engineering and science [56, 57]. 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.

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 $\Delta t \rightarrow 0$. During each time interval Δt, a faulty component i infects its neighboring components with probability β. The bigger the probability β is for highly connected components, the greater the average time to full failure propagation.

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 (3). This makes the large design problems solvable with closed-form solutions. Notice that the *independence assumption* is theoretically very close to the full Markov chain and does not place any constraints on the design network topology. In addition, the NLDS model is design based on the assumption that the failure cascading rate is the same for all the components. The reason for this is that in this paper, each sub(sustem), i.e. electronic network, use similar links between the components which propagate the failure with similar rate. Therefore in this specific case, different domains are modeled as different sub(network) with the same value for . Future work will consider different failure propagation rate for different components.

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 ($\Delta t \rightarrow 0$), the possibility of multiple cascades within the same Δt is small and can be ignored.

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

$$= \prod_{j:\, neighbor\ of\ i} (1-\beta * p_j(t-1)) \tag{2}$$

In the above formula (1), it is assumed that $p_j(t-1)$ are independent from one another.
As illustrated in Fig. 7, each component at time-step t, is either Nominal (***N***) or Failed (***F***). A nominal component ***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*.

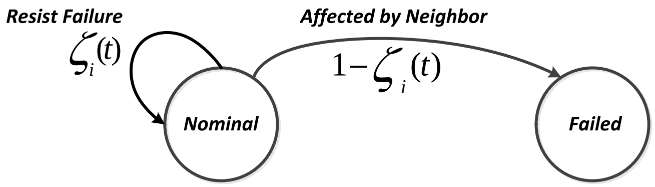


Fig. 7: Transition Diagram of the Nominal-Failed (NF) Model.

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

$$1 - p_i(t) = (1 - p_i(t-1))\zeta_i(t) \quad i = 1...N \tag{3}$$

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:

$$\eta_t = \sum_{i=1}^{N} p_i(t) \tag{4}$$

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. The value of λ and μ are chosen based on expert knowledge and historical failure and repair data [58–60]. 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:

$$\frac{dS}{dt} = -\lambda\bar{k}\rho S \tag{5}$$

$$\frac{d\rho}{dt} = -\mu\rho + \lambda\bar{k}\rho S \tag{6}$$

$$\frac{dF}{dt} = \mu\rho \tag{7}$$

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 ρ*(0)* $\simeq$ *0* (small number of faulty components exist in the initial design) is assumed. Therefore the following can be obtained for *S(t)*:

$$S(t) = e^{-\lambda\bar{k}\rho F(t)} \tag{8}$$

4 Case Study

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 (8) can be replaced by the following:

$$S_k(t) + \rho_k(t) + F_k(t) = 1 \tag{9}$$

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.

As depicted in Fig. 8, a quad-redundant Electro-Mechanical Actuator (EMA) [61] 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. 9, 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. 9, to move the aircraft FCS to the command positions.

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 must take on the full load from the FCS in the extreme case 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 dead-weights. Once an EMA is disengaged from the system it cannot be re-engaged automatically. It

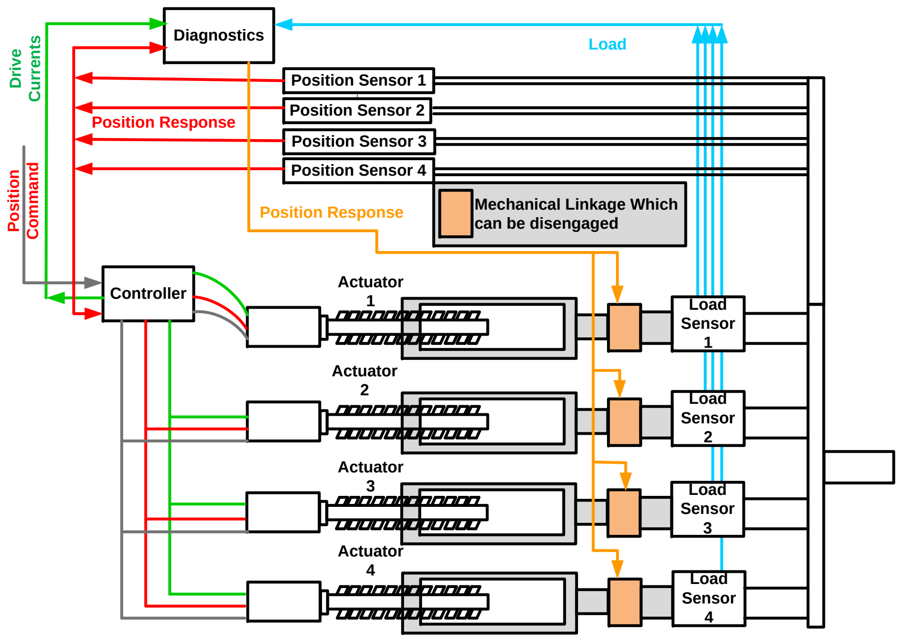


Fig. 8: Quad-Redundant EMA Scheme.

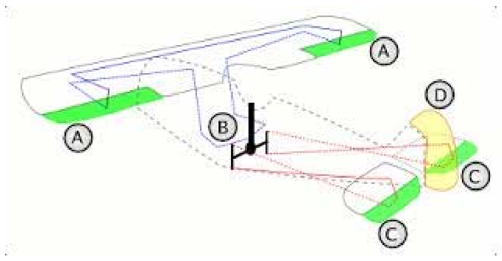


Fig. 9: Basic Aircraft Control Surfaces.

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. 8. 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.

4.1 Safety Requirements Modeling Using SysML

In the case study of Fig. 9, 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. Fig. 10 depicts a set of high-level requirements. To facilitate the verification process, each level of requirements are associated with a formal Finite State Process (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, including sub-systems and components to map each requirement to a traceable source. As depicted in Fig. 5, requirements mapping are made possible by using the *satisfy* relationship to link a single or set of blocks to one or more requirements.

4.2 Automated Assume-guarantee Reasoning

In order to transform the requirements and the design architecture into a finite model, we use finite labelled transition systems (LTS). 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. 11 represents this model graphically. State *0* corre-

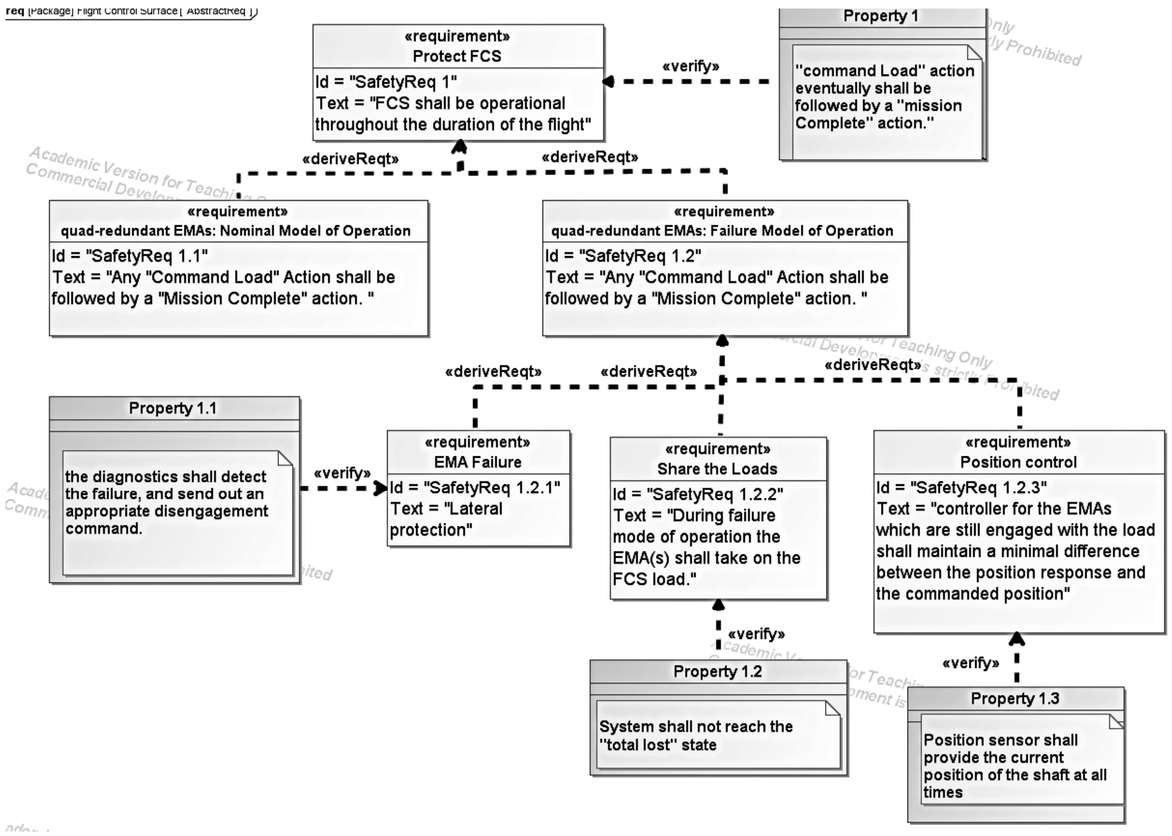

req [Package] Flight Control Surface [AbstractReq]
«requirement»
Protect FCS
Id = "SafetyReq 1"
Text = "FCS shall be operational throughout the duration of the flight"
«verify»
Property 1
"command Load" action eventually shall be followed by a "mission Complete" action."
«deriveReqt»
«deriveReqt»
«requirement»
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

Fig. 10: Quad-redundant EMAs High-Level Requirements.

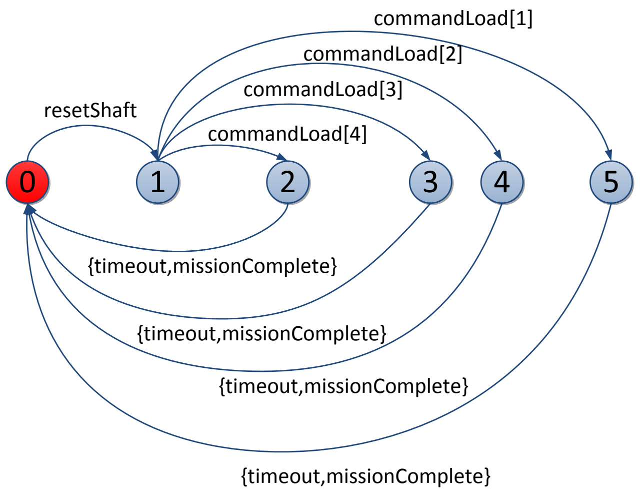

commandLoad[1]
commandLoad[2]
commandLoad[3]
commandLoad[4]
resetShaft
0
1
2
3
4
5
{timeout,missionComplete}
{timeout,missionComplete}
{timeout,missionComplete}
{timeout,missionComplete}

Fig. 11: LTS Model of the Command Unit Subsystem.

sponds 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.

The labelled transition systems (LTS) *CommandUnit* represented in Fig. 11 can be expressed in FSP as shown in Table 1.

Table 1: 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 2, 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 *controller* performs action <*setLoad*[*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

Table 2: 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]).**

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.

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 3 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.

Table 3: 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).**

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 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 4: Parallel Composition of EMA (Table 3) and Controller (Table 2)

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 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 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

Table 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	–

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.

So far, we provided the basis for decomposing and modeling the system based on the modular description of the design components and subsystems. Next, 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 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: FSP Description of The Safety Requirement

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

The $<safeOpn>$ property of Table 6, 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. 12, while translating the FSP notation of a property, 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. 12 is recognized as an *error* LTS with the failure state of *-1*.

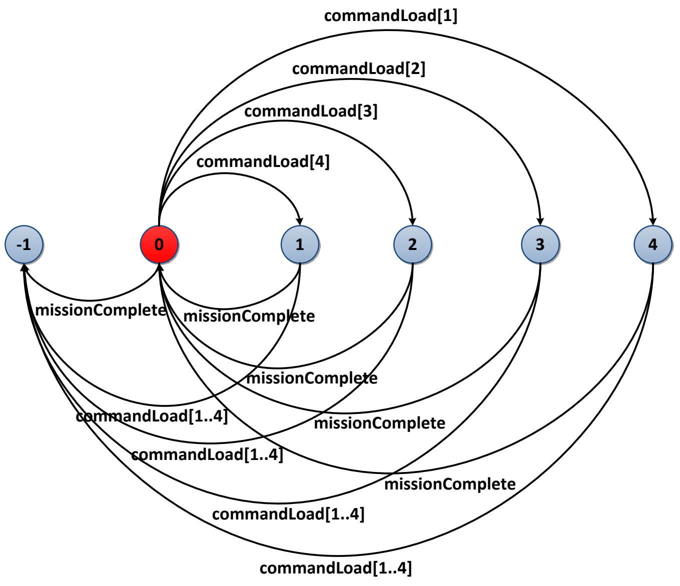


Fig. 12: The LTS Model of the Safety Operation Property.

In the case of the $<SafeOpn>$ property, the verification algorithm had detected a property violation. Table. 7 represents the sequence of actions that lead to the failure state.

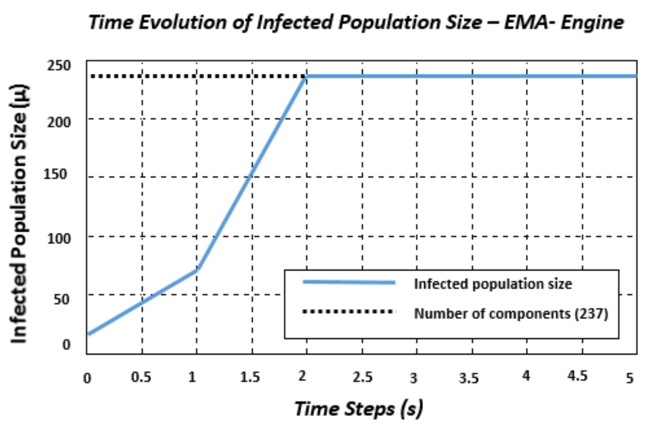
Table 7: Trace To Property Violation

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

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 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.

4.3 Design Topology and its Effect on Failure Propagation

In order to model the propagation characteristics of failures in complex engineered systems, two failure propagation models are used. The quad-redundant Electro Mechanical Actuator (EMA) 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



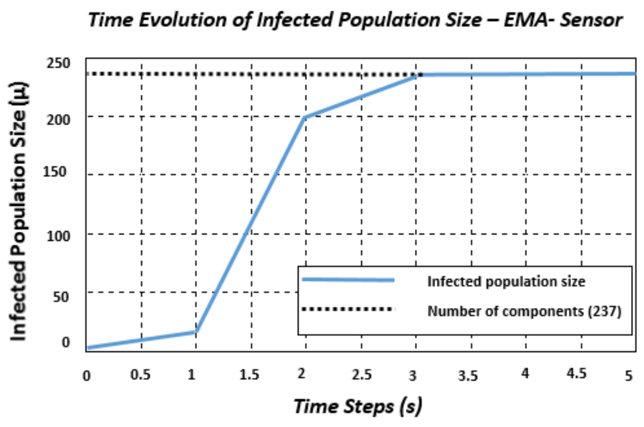
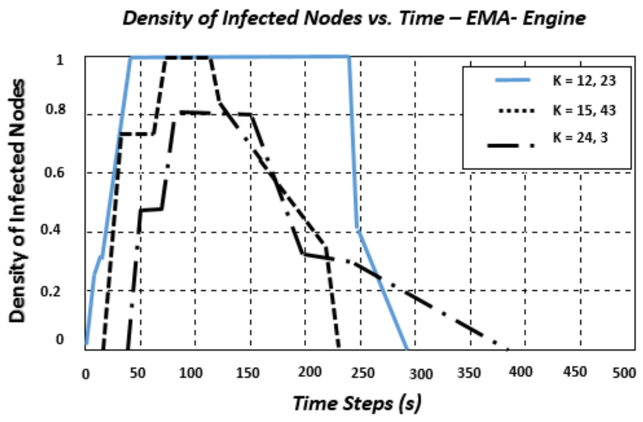


Fig. 13: Time Evolution of Faulty Components' Population Size (Origin of Failure: (Left Picture: EMA Engine) and (Right Picture: Sensor)).



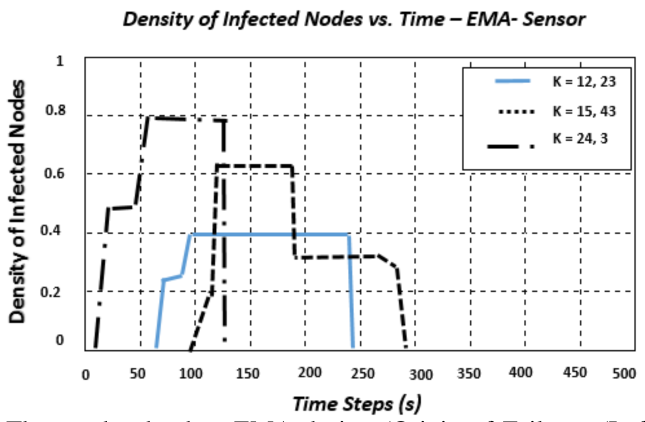


Fig. 14: Time Evolution of Faulty Components Density for The quad-redundant EMA design (Origin of Failure: (Left Picture: EMA Engine) and (Right Picture: Sensor)).

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 can be seen in Fig. 13 the population of the infected components with respect to time is different for the two experiments. In the left hand side of Fig. 13, 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 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 λ = 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. 14 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. 14, 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 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 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. Three most representative sets in each figure is chosen and drawn.

The left hand side of Fig. 14, 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.

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. 15 <*Fault_Tolerance*> property contains {<commandLoad[1..4],d[1..4].jam, and missionComplete>} actions. The property of Table 8, 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 8: 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] → Complete-
Mission[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]).
```

As can be seen in the compositional model of Table 9, 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 9 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 the property is instantiated allowing four

Table 9: 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).
```

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

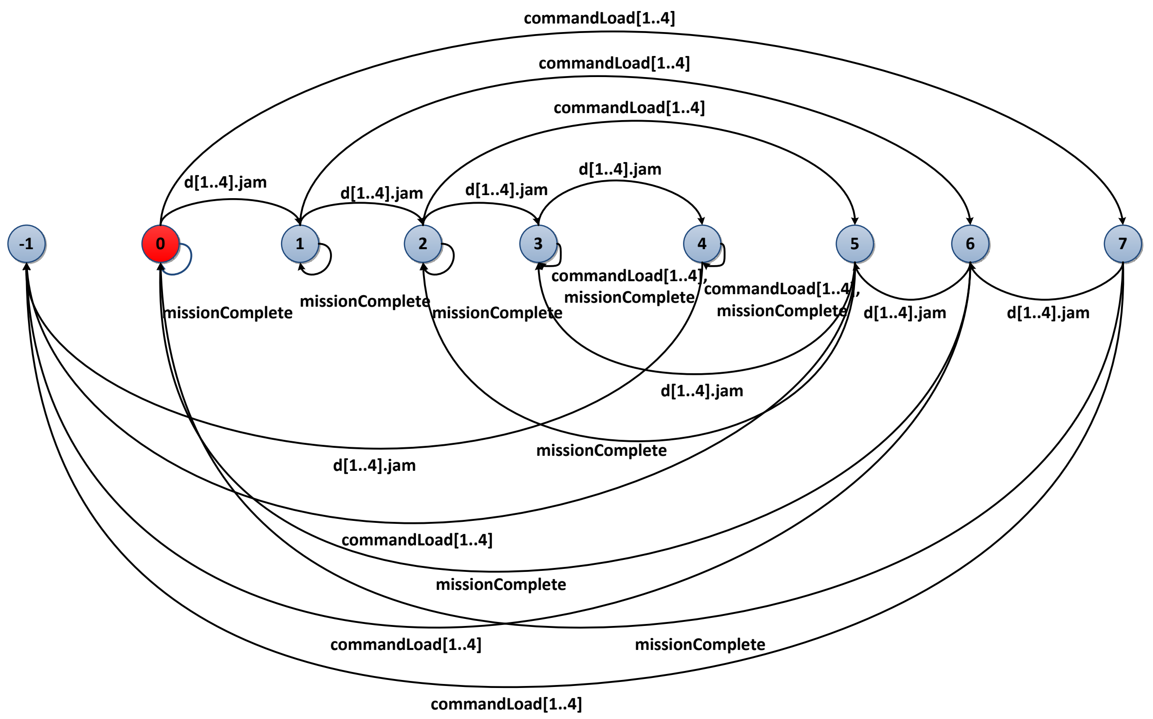


Fig. 15: LTS Model Of The Fault Tolerance Property.

EMAs to fail, the safety analysis verifies that the property is violated and a failure propagation path similar to the one in Table 7 is produced. Therefore, the generic safety property modeled in Table 8 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.

5 Discussion

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 of 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.

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. The NLDS model adequately identified 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. Both models provide insight into design architectures that can be more resilient to failures.

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 Conclusion and Future Work

In this research, resilience is addressed from different viewpoints and a framework is presented that jointly 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 automated assume-guarantee reasoning technique. Secondly, resilience design is characterized as part of the physical infrastructure of the design that is robust and has the ability to survive disruptions. Lastly, through the resilience design approach, system is 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 of 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 a set of tools to support them to integrate safety and resilience where 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 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 behavior for systems, based on design architecture. The proposed technique enables system designers to assess failure behavior from the analysis of components of the system, however, these algorithms have to expanded so they can assess the probability of system-level failures based on failures of components. This approach should be connected to a probabilistic model checker, which will allow verification of the failure models.

References

[1] Chase, K. W., and Parkinson, A. R., 1991. "A survey of research in the application of tolerance analysis to the design of mechanical assemblies". *Research in Engineering design,* **3**(1), pp. 23–37.

[2] Martin, M. V., and Ishii, K., 2002. "Design for variety: developing standardized and modularized product platform architectures". *Research in Engineering Design,* **13**(4), pp. 213–235.

[3] Ameri, F., Summers, J. D., Mocko, G. M., and Porter, M., 2008. "Engineering design complexity: an investigation of methods and measures". *Research in Engineering Design,* **19**(2-3), pp. 161–179.

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

[5] Bedau, M. A., 2002. "Downward causation and the autonomy of weak emergence". *Principia,* **6**(1), pp. 5–50.

[6] Chalmers, D. J., 2002. "Varieties of emergence". *Department of Philosophy, University of Arizona, USA, Tech. rep./preprint*.

[7] Seager, W., 2012. "Emergence and supervenience". In *Natural Fabrications*. Springer, pp. 121–154.

[8] Pavard, B., Dugdale, J., Saoud, N. B.-B., Darcy, S., and Salembier, P., 2006. "Design of robust sociotechnical systems". *Resilience engineering, Juan les Pins, France*.

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

[10] Leveson, N., 2004. "A new accident model for engineering safer systems". *Safety science,* **42**(4), pp. 237–270.

[11] Hollnagel, E., Woods, D. D., and Leveson, N., 2007. *Resilience engineering: Concepts and precepts*. Ashgate Publishing, Ltd.

[12] Čepin, M., 2011. "Reliability block diagram". In *Assessment of Power System Reliability*. Springer, pp. 119–123.

[13] Do, D., 1980. "Procedure for performing a failure mode, effects and criticality analysis". *Department of Defense, Washington DC*.

[14] Stone, R. B., Tumer, I. Y., and Van Wie, M., 2005. "The Function-Failure Design Method". *Journal of Mechanical Design(Transactions of the ASME),* **127**(3), pp. 397–407.

[15] Tumer, I. Y., Stone, R. B., and Bell, D. G., 2003. "Requirements for a Failure Mode Taxonomy for Use in Conceptual Design". In Proceedings of the International Conference on Engineering Design, ICED, Vol. 3, Paper 1612, Stockholm,, Sweden.

[16] Krus, D., and Lough, K. G., 2007. "Applying Function-Based Failure Propagation in Conceptual Design". ASME.

[17] Lough, K. G., Stone, R. B., and Tumer, I., 2006. "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.

[18] Devendorf, E., and Lewis, K., 2008. "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, American Society of Mechanical Engineers, pp. 273–284.

[19] Kurtoglu, T., Tumer, I. Y., and Jensen, D. C., 2010. "A Functional Failure Reasoning Methodology for Evalu-

ation of Conceptual System Architectures". *Research in Engineering Design,* **21**(4), pp. 209–234.

[20] Lee, W.-S., Grosh, D., Tillman, F. A., and Lie, C. H., 1985. "Fault Tree Analysis, Methods, and Applications? A Review". *Reliability, IEEE Transactions on,* **34**(3), pp. 194–203.

[21] Ericson, C. A., 2005. "Event tree analysis". *Hazard Analysis Techniques for System Safety*, pp. 223–234.

[22] Greenfield, M. A., 2001. "Nasa's use of quantitative risk assessment for safety upgrades". *Space safety, rescue and quality 1999-2000*, pp. 153–159.

[23] Stamatelatos, M., Dezfuli, H., Apostolakis, G., Everline, C., Guarro, S., Mathias, D., Mosleh, A., Paulos, T., Riha, D., Smith, C., et al., 2011. "Probabilistic risk assessment procedures guide for nasa managers and practitioners".

[24] Venkatasubramanian, V., Zhao, J., and Viswanathan, S., 2000. "Intelligent systems for hazop analysis of complex process plants". *Computers & Chemical Engineering,* **24**(9), pp. 2291–2302.

[25] Hollnagel, E., 2004. *Barriers and accident prevention*. Ashgate Publishing, Ltd.

[26] Johnson, C. W., and Holloway, C. M., 2003. "The esa/-nasa soho mission interruption: Using the stamp accident analysis technique for a software related mishap". *Software: Practice and Experience,* **33**(12), pp. 1177–1198.

[27] Berezin, S., Campos, S., and Clarke, E. M., 1998. *Compositional reasoning in model checking*. Springer.

[28] Alur, R., Madhusudan, P., and Nam, W., 2005. "Symbolic Compositional Verification By Learning Assumptions". In Computer Aided Verification, Springer, pp. 548–562.

[29] Cobleigh, J. M., Giannakopoulou, D., and Păsăreanu, C. S., 2003. "Learning Assumptions For Compositional Verification". In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, pp. 331–346.

[30] Giannakopoulou, D., Păsăreanu, C. S., and Barringer, H., 2005. "Component Verification With Automatically Generated Assumptions". *Automated Software Engineering,* **12**(3), pp. 297–320.

[31] Nam, W., and Alur, R., 2006. "Learning-based Symbolic Assume-guarantee Reasoning With Automatic decomposition". In *Automated Technology for Verification and Analysis*. Springer, pp. 170–185.

[32] Chaki, S., Clarke, E., Sinha, N., and Thati, P., 2005. "Automated Assume-guarantee Reasoning For Simulation Conformance". In Computer Aided Verification, Springer, pp. 534–547.

[33] Mehrpouyan, H., Giannakopoulou, D., Brat, G., Tumer, I. Y., and Hoyle, C., 2013. "Complex system design verification using assumption generation". In ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers.

[34] Ullman, D. G., 2002. "Toward the ideal mechanical engineering design support system". *Research in Engineering Design,* **13**(2), pp. 55–64.

[35] Booch, G., Rumbaugh, J., and Jacobson, I., 1999. *The unified modeling language user guide*. Pearson Education India.

[36] *Rational DOORS*.

[37] Geensoft, R. Effective solution for managing requirements traceability and impact analysis across hardware and software projects lifecycle.

[38] Friedenthal, S., Moore, A., and Steiner, R., 2011. *A practical guide to SysML: the systems modeling language*. Elsevier.

[39] Blaise, J.-C., Lhoste, P., and Ciccotelli, J., 2003. "Formalisation of normative knowledge for safe design". *Safety Science,* **41**(2), pp. 241–261.

[40] Madni, A., 2007. "Designing for resilience". *ISTI Lecture Notes on Advanced Topics in Systems Engineering*.

[41] Henzinger, T., and Sifakis, J., 2006. "The Embedded Systems Design Challenge". pp. 1–15.

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

[43] Giannakopoulou, D., Pasareanu, C. S., and Barringer, H., 2002. "Assumption Generation for Software Component Verification". In Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on, IEEE, pp. 3–12.

[44] Cobleigh, J., Giannakopoulou, D., and Păsăreanu, C., 2003. "Learning Assumptions for Compositional Verification". *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 331–346.

[45] Cheung, S. C., and Kramer, J., 1999. "Checking Safety Properties Using Compositional Reachability Analysis". *ACM Transactions on Software Engineering and Methodology (TOSEM),* **8**(1), pp. 49–78.

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

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

[48] Rodrigues, R. W., 2000. "Formalising UML Activity Diagrams Using Finite State Processes". In Proc. of the 3rd Intl. Conf. on the Unified Modeling Language, York, UK, Citeseer.

[49] Mehrpouyan, H., Haley, B., Dong, A., Tumer, I. Y., and Hoyle, C., 2013. "Resilient design of complex engineered systems". In ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. V03AT03A048–V03AT03A048.

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

[51] Michelena, N. F., and Papalambros, P. Y., 1997. "A hypergraph framework for optimal model-based decomposition of design problems". *Computational Optimization and Applications,* **8**(2), pp. 173–196.

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

[53] Braha, D., and Reich, Y., 2003. "Topological structures for modeling engineering design processes". *Research in Engineering Design,* **14**(4), pp. 185–199.

[54] Palmer, R. S., and Shapiro, V., 1993. "Chain models of physical behavior for engineering analysis and design". *Research in Engineering Design,* **5**(3-4), pp. 161–184.

[55] Harris, T. E., 1974. "Contact interactions on a lattice". *The Annals of Probability*, pp. 969–988.

[56] Marro, J., and Dickman, R., 2005. *Nonequilibrium Phase Transitions in Lattice Models*. Cambridge University Press.

[57] Durrett, R., 1999. "Stochastic Spatial Models". *Siam Review,* **41**(4), pp. 677–718.

[58] Bodden, D. S., Clements, N. S., Schley, B., and Jenney, G., 2007. "Seeded failure testing and analysis of an electro-mechanical actuator". In Aerospace Conference, 2007 IEEE, IEEE, pp. 1–8.

[59] Aten, M., Towers, G., Whitley, C., Wheeler, P., Clare, J., and Bradley, K., 2006. "Reliability comparison of matrix and other converter topologies". *Aerospace and Electronic Systems, IEEE Transactions on,* **42**(3), pp. 867–875.

[60] Raimondi, G., McFarlane, R., Bingham, C., Atallah, K., Howe, D., Mellor, P., Capewell, R., and Whitley, C., 1998. "Large electromechanical actuation systems for flight control surfaces".

[61] Balaban, E., Saxena, A., Goebel, K., Byington, C., Watson, M., Bharadwaj, S., Smith, M., and Amin, S., 2009. "Experimental Data Collection And Modeling For Nominal And Fault Conditions On Electromechanical Actuators". In Annual Conference of the Prognostics and Health Management Society, pp. 1–15.