Formal Verification of Complex Systems based on SysML Functional Requirements

Hoda Mehrpouyan[1], Irem Y. Tumer[2], Chris Hoyle[2], Dimitra Giannakopoulou[3], Guillaume Brat[3]

[1] *TSYS School of Computer Science, Columbus State University, Columbus, GA, USA*
mehrpouyan_hoda@columbusstate.edu

[2] *School of Mechanical, Industrial, and Manufacturing Engineering, Oregon State University, Corvallis, OR, USA*
irem.tumer@oregonstate.edu, chris.hoyle@oregonstate.edu

[3] *NASA Ames Research Center, Moffett Field, CA, USA*
dimitra.Giannakopoulou@nasa.gov, guillaume.p.brat@nasa.gov

ABSTRACT

As modern systems continue to increase in size and complexity, they pose increasingly significant safety and risk management challenges. A model-based safety approach is an efficient way of coping with the increasing system complexity. It helps better manage the complexity by utilizing reasoning tools that require abstract models to detect failures as early as possible during the design process. This paper develops 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, and an *assume-guarantee* technique 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.

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 even more important. Moreover, the growing complexity of such systems has made it more challenging to achieve design solutions that satisfy safety and reliability requirements (Wiese & John, 2003; Zio, 2009; N. Leveson, 2011). Hollnagel et al. (Hollnagel, Woods, & Leveson, 2007) 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. (Baroth et al., 2001) recommends the Prognostic and Health Management System (PHMS) 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 (N. G. Leveson, 2009). 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 (IEC, 1998) and the SAE & EUROCAE standards in the avionic industry (ARP4761, 1996; ARP4754, 1996). However, these standards do not specify how to implement formal approaches throughout the design process.

Strategies for engineered system design emerge from a process of requirement decomposition and transforming requirement models into the conceptual models (Blanchard, 2012; Buede, 2011). 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 (Hirtz, Stone, McAdams, Szykman, & Wood, 2002; Nagel, Stone, Hutcheson, McAdams, & Donndelinger, 2008; Kurtoglu & Campbell, 2009) 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:

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

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.

2. RELATED WORK

Different standards, e.g., (IEEE1220, 2005; ISO-IEC15288, 2002) 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. (Lundteigen, Rausand, & Utne, 2009) 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.

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) (OMG, 2007) to support various aspect of system modeling, Rational Doors (IBM, 2010) to express the requirements, and Reqtify (GeenSys, 2008) 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. SysML focuses on the global features of architectural views, whereas other modeling languages such as he Architecture Analysis and Design language (AADL) addresses the more detailed platform-oriented and physical aspects of such systems. 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.

2.2. Model Checking and Formal Verification

Model checking is one of the approaches to formal verification of finite state hardware and software systems (Henzinger,

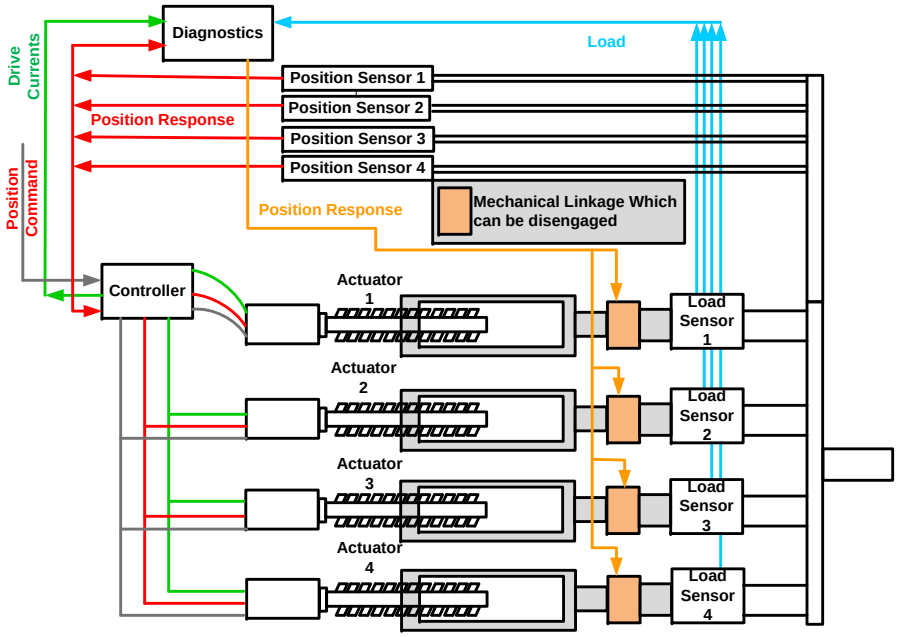

Figure 1. Quad-Redundant EMA Scheme.

Ho, & Wong-Toi, 1997; Henzinger, Nicollin, Sifakis, & Yovine, 1994). 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. (Barragan, Roth, Faure, et al., 2006) 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 involving 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 (Evrot, Petin, & Mery, 2006). 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 (Tumer, Stone, & Bell, 2003; Stone, Tumer, & Stock, 2005; Kurtoglu & Tumer, 2008; Tumer & Smidts, 2011), this paper aims at addressing this challenge using the system-oriented SysML-based modeling approach combined with formal verification technique.

2.3. Case Study

As depicted in Fig. 1, a quad-redundant Electro-Mechanical Actuator (EMA) (Balaban et al., 2009) 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. 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. 2, to move the aircraft FCS to the command positions.

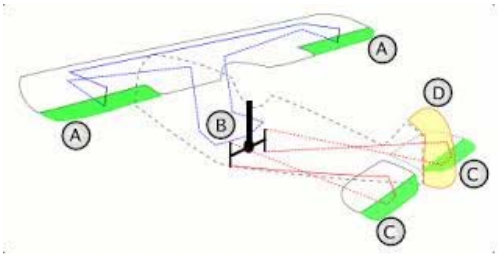

Figure 2. Basic Aircraft Control Surfaces.

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

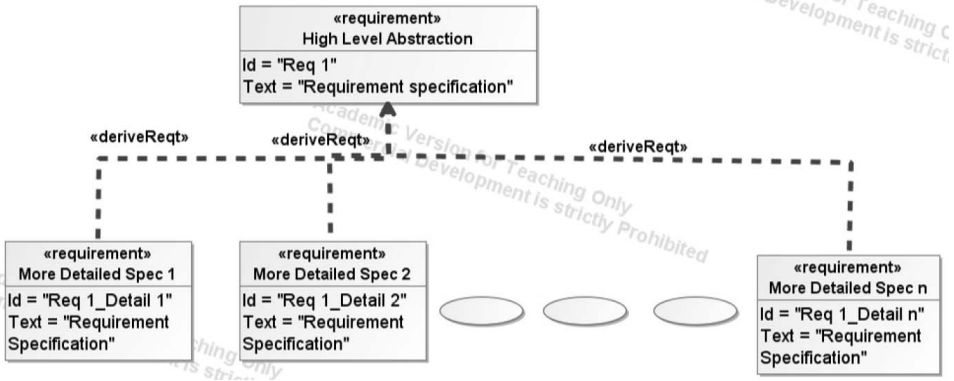

Figure 3. Requirements Decomposition.

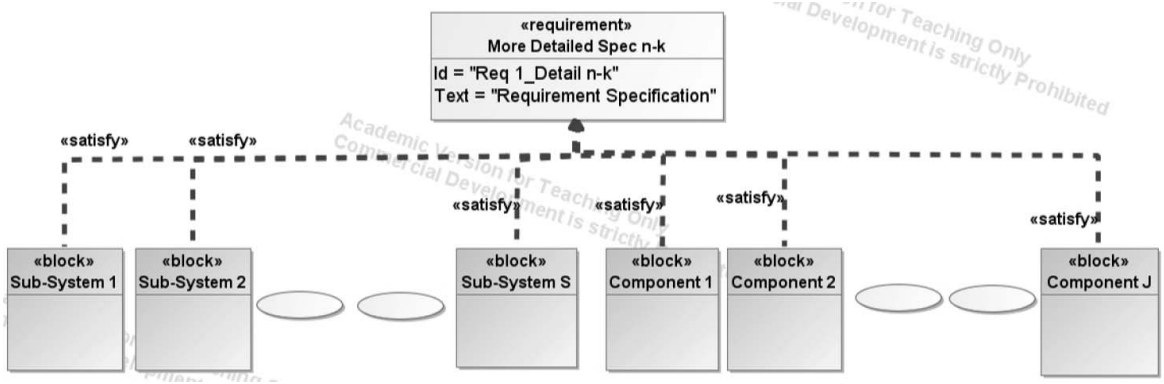

Figure 4. Requirements Mapping.

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 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. 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. 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.

3.1. 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 depicts the decomposition of a single abstract requirement into several more explicit ones. A study by Blaise et al. (Blaise, Lhoste, & Ciccotelli, 2003) 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. 4 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. (Madni, 2007) 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. (Hollnagel et al., 2007) 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. 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.

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

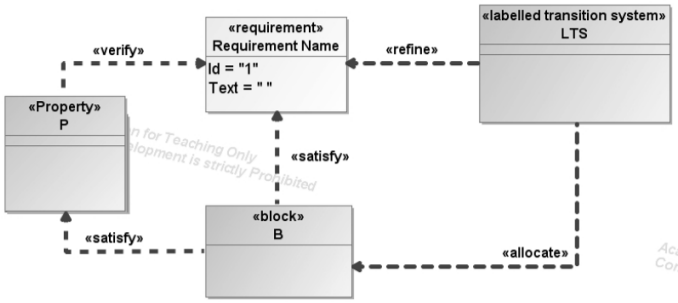

Figure 5. Requirements Traceability.

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. 5).

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* (Cobleigh, Giannakopoulou, & Păsăreanu, 2003; Giannakopoulou, Păsăreanu, & Barringer, 2005; Nam & Alur, 2006; Chaki, Clarke, Sinha, & Thati, 2005) 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) (Rodrigues, 2000) 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. 1 is provided in Table 1 and Fig. 6 respectively.

Table 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).
```

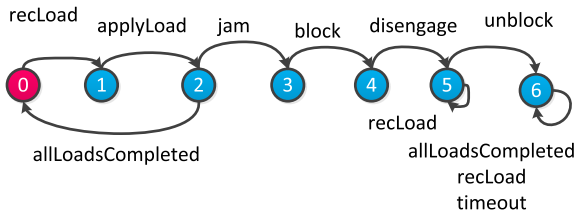

Figure 6. LTS Model of the EMA Subsystem.

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 6 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.

After system modeling, the actual analysis of the models is carried out utilizing the Assume Guarantee Reasoning (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 (Henzinger, Qadeer, & Rajamani, 1998; Chaki et al., 2005):

Rule ASYM

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

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

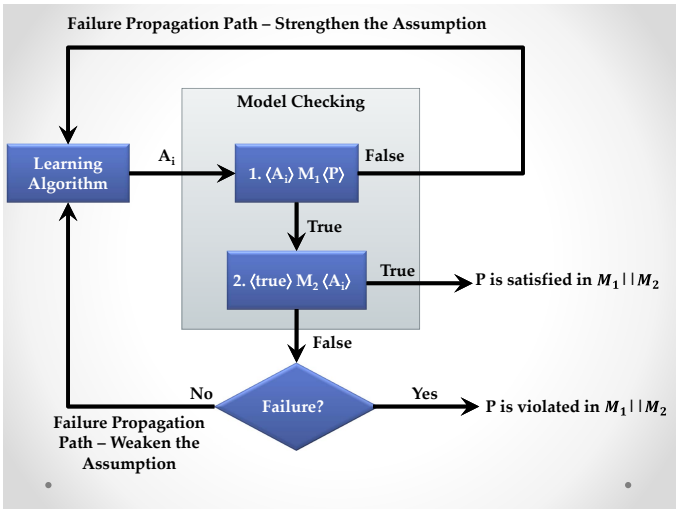

Figure 7. An Overview of the Algorithm that Generates Assumptions.

In this research, the algorithm in (Giannakopoulou, Pasareanu, & Cobleigh, 2004) 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 framework of Figure 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 $\langle \mathit{true}\rangle\ M_2\ \langle A\rangle$. If the rule still holds, then it is concluded that $\langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle$. 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 $\langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle$ violates the property P.

4. APPLICATION ON THE CASE STUDY

In the case study of Fig. 2, the Flight Control Surface (FCS) must meet rigorous safety and availability requirements be-

Table 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

fore 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. 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. 9 depicts a set of high-level requirements. To facilitate the verification process, each level of requirements are associated with a formal FSP using property stereotype in SysML. Therefore, satisfying a 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. 8), including sub-systems and components to map each requirement to a traceable source. As depicted in Fig. 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. 2.
In order to transform the requirements and the design architecture presented in Fig. 8 into a finite model, we use 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 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, 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 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]).
```

The partial LTS model of the controller is depicted in Fig. 10. 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.

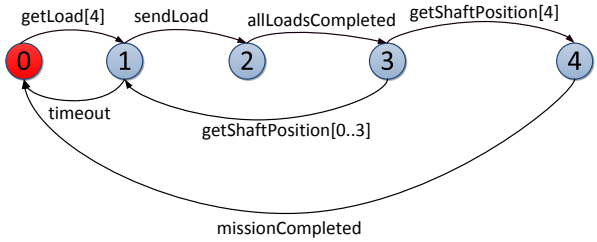

Figure 10. LTS Model of the Controller Subsystem.

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 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

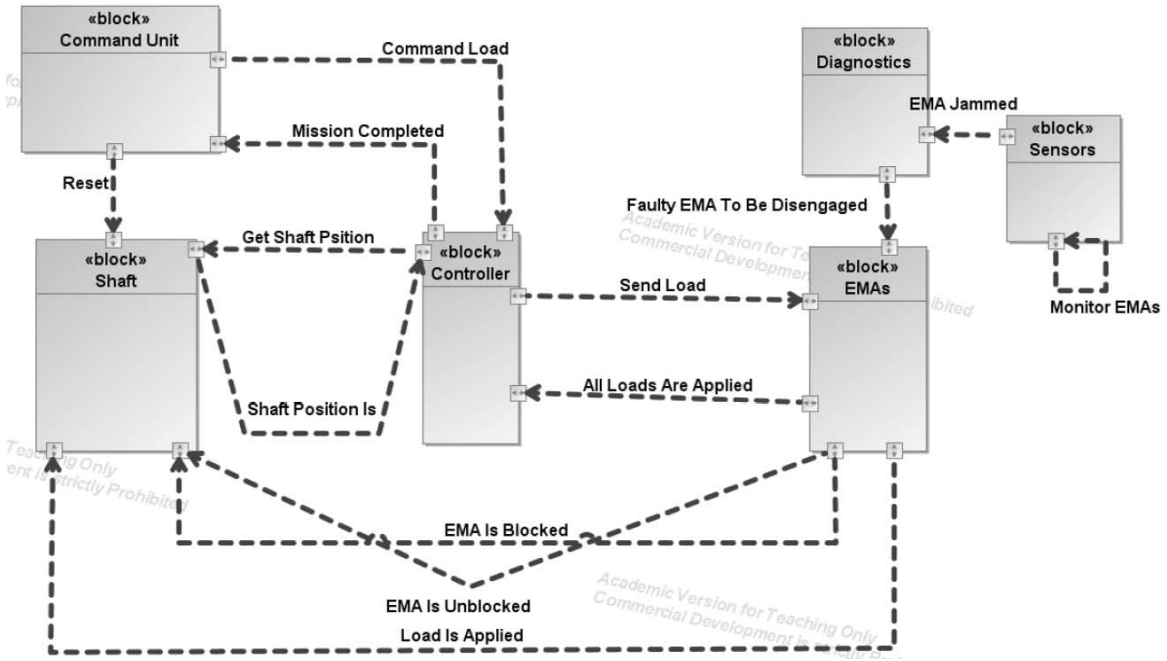

Figure 8. Structural Model of the Quad-redundant EMAs.

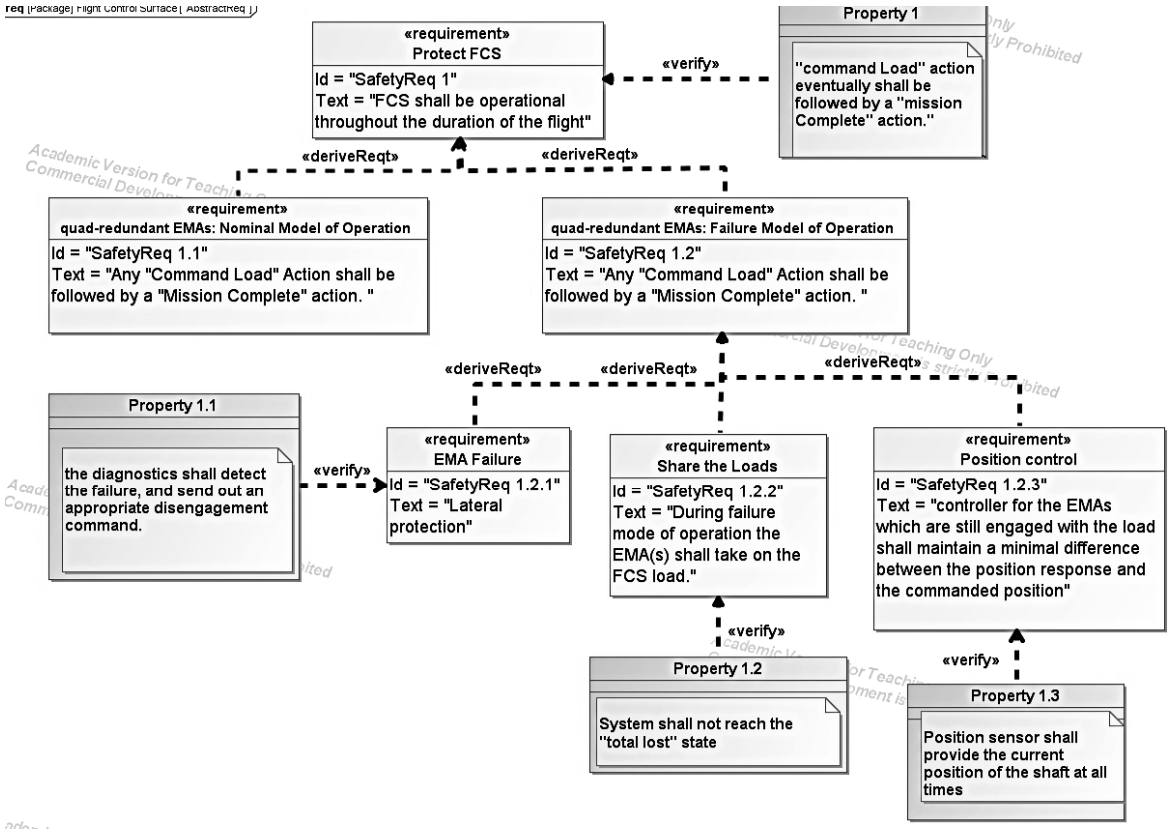

Figure 9. Quad-redundant EMAs High-Level Requirements.

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 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

Table 4. Parallel Composition of EMA (Table 1) and Controller (Table 3)

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

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

Table 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. 7 verifies that the safety property is satisfied.

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 6 verifies that the system never reaches the failure condition of *total loss* if and only if *N ≤ 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.

5. CONCLUSION

There is a growing demand for formal methods and tools that facilitate the specification and verification of complex engi-

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

neered 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.

REFERENCES

ARP4754, S. (1996). Certification considerations for highly-integrated or complex aircraft systems. *Society of Automotive Engineers Inc*.

ARP4761, S. (1996). Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *SAE International, December*.

Balaban, E., Saxena, A., Goebel, K., Byington, C., Watson, M., Bharadwaj, S., ... Amin, S. (2009). Experimental Data Collection And Modeling For Nominal And Fault Conditions On Electro-mechanical Actuators. In *Annual conference of the prognostics and health management society* (pp. 1–15).

Baroth, E., Zakrajsek, J., Powers, W., Fox, J., Prosser, B., Pallix, J., & Schweikard, K. (2001). Ivhm (Integrated Vehicle Health Management) techniques for future space vehicles. In *37th joint propulsion conference.*

Barragan, I. S., Roth, M., Faure, J.-M., et al. (2006). 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.*

Blaise, J.-C., Lhoste, P., & Ciccotelli, J. (2003). Formalisation of normative knowledge for safe design. *Safety Science*, *41*(2), 241–261.

Blanchard, B. S. (2012). *System engineering management* (Vol. 64). Wiley. com.

Buede, D. M. (2011). *The engineering design of systems: Models and methods* (Vol. 55). John Wiley & Sons.

Chaki, S., Clarke, E., Sinha, N., & Thati, P. (2005). Automated Assume-guarantee Reasoning for Simulation Conformance. In *Computer aided verification* (pp. 241–246).

Cobleigh, J. M., Giannakopoulou, D., & Păsăreanu, C. S. (2003). Learning Assumptions For Compositional Verification. In *Tools and algorithms for the construction and analysis of systems* (pp. 331–346). Springer.

Evrot, D., Petin, J.-F., & Mery, D. (2006). Formal specification of safe manufacturing machines using the b method: Application to a mechanical. In *Information control problems in manufacturing* (Vol. 12, pp. 281–286).

GeenSys. (2008). *Reqtify.* www.geensys.com.

Giannakopoulou, D., Păsăreanu, C. S., & Barringer, H. (2005). Component Verification With Automatically Generated Assumptions. *Automated Software Engineering*, *12*(3), 297–320.

Giannakopoulou, D., Pasareanu, C. S., & Cobleigh, J. M. (2004). Assume-guarantee verification of source code with design-level assumptions. In *Proceedings of the 26th international conference on software engineering* (pp. 211–220).

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

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

Henzinger, T. A., Qadeer, S., & Rajamani, S. K. (1998). You assume, we guarantee: Methodology and case studies. In *Computer aided verification* (pp. 440–451).

Hirtz, J., Stone, R. B., McAdams, D. A., Szykman, S., & Wood, K. L. (2002). A Functional Basis For Engineering Design: Reconciling And Evolving Previous Efforts. *Research in engineering Design*, *13*(2), 65–82.

Hollnagel, E., Woods, D. D., & Leveson, N. (2007). *Resilience engineering: Concepts and precepts*. Ashgate Publishing, Ltd.

IBM. (2010). *Rational doors.* Available from: http://www-01.ibm.com/software/awdtools/doors.

IEC. (1998). 61508 functional safety of electrical/electronic/programmable electronic safety-related systems. *International electrotechnical commission.*

IEEE1220. (2005). *IEEE standard for application and man-*

agement of the systems engineering process. IEEE New York, NY, USA.

ISO-IEC15288. (2002). *Systems engineering system life cycle processes.* International Standardization Organization.

Kurtoglu, T., & Campbell, M. I. (2009). Automated Synthesis Of Electromechanical Design Configurations From Empirical Analysis Of Function To Form Mapping. *Journal of Engineering Design*, *20*(1), 83–104.

Kurtoglu, T., & Tumer, I. Y. (2008). A graph-based fault identification and propagation framework for functional design of complex systems. *Journal of Mechanical Design*, *130*(5), 051401.

Leveson, N. (2011). *Engineering a safer world: Systems thinking applied to safety*. MIT Press.

Leveson, N. G. (2009). The need for new paradigms in safety engineering. In *Safety-critical systems: Problems, process and practice* (pp. 3–20). Springer.

Lundteigen, M. A., Rausand, M., & Utne, I. B. (2009). Integrating rams engineering and management with the safety life cycle of iec 61508. *Reliability Engineering & System Safety*, *94*(12), 1894–1903.

Madni, A. (2007). Designing for resilience. *ISTI Lecture Notes on Advanced Topics in Systems Engineering*.

Nagel, R. L., Stone, R. B., Hutcheson, R. S., McAdams, D. A., & Donndelinger, J. A. (2008). 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)* (pp. 273–286).

Nam, W., & Alur, R. (2006). Learning-based Symbolic Assume-guarantee Reasoning With Automatic decomposition. In *Automated technology for verification and analysis* (pp. 170–185). Springer.

OMG, O. (2007). *Unified modeling language (omg uml).* Superstructure.

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.*

Stone, R. B., Tumer, I. Y., & Stock, M. E. (2005). Linking product functionality to historic failures to improve failure analysis in design. *Research in Engineering Design*, *16*(1-2), 96–108.

Tumer, I. Y., & Smidts, C. S. (2011). Integrated design-stage failure analysis of software-driven hardware systems. *Computers, IEEE Transactions on*, *60*(8), 1072–1084.

Tumer, I. Y., Stone, R. B., & 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).

Wiese, P. R., & John, P. (2003). *Engineering design in the multi-discipline era: A systems approach*. Wiley.

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

BIOGRAPHIES

Hoda Mehrpouyan Dr. Hoda Mehrpouyan is a Professor in TSYS School of Computer Science at Columbus State University. Her research focuses on model-based systems engineering, resilience and safety analysis, information technology, simulation and verification to support the design of complex systems. She received her Ph.D. from Oregon State University in 2014 and holds a M.S. degree in Software Engineering from Linköping University in 2011. Prior to returnung to academia, She spent 7 years in industry as a system delivery consultant and programmer analyst.

Irem Tumer Dr. Irem Y. Tumer is a Professor in Mechanical, Industrial, and Manufacturing Engineering at Oregon State University, where she leads the Complex Engineered System Design Laboratory, and Associate Dean for Research and Economic Development for the College of Engineering at OSU. Her research focuses on the overall problem of designing highly complex and integrated engineering systems with reduced risk of failures, and developing formal methodologies and approaches for complex system design and analysis. Since moving to Oregon State University in 2006, her funding has largely been through NSF, AFOSR, DARPA, and NASA. Prior to accepting a faculty position at OSU, Dr. Tumer led the Complex Systems Design and Engineering group in the Intelligent Systems Division at NASA Ames Research Center, where she worked from 1998 through 2006 as Research Scientist, Group Lead, and Program Manager. Dr. Tumer has been Conference Chair for ASME Design for Manufacturing and the Lifecycle conference in 2000, Program Chair for IEEE Reliability Society?s Prognostics and Health Management Conference in 2008, and Program Chair (2011) and Conference Chair (2012) for ASME?s International Design Theory and Methodology Conference; and is current Associate Editor for ASME Journal of Mechanical Design and the International Journal of Prognostics and Health Management, and guest editor for AEIDAM journal. She received her Ph.D. in Mechanical Engineering from The University of Texas at Austin in 1998.

Christopher Hoyle Dr. Hoyle's research focuses on decision making in engineering design, with emphasis on the early design phase when uncertainty is high and the potential design space is large. More specifically, he works in the areas of decision-based design (linking consumer preferences and enterprise-level objectives with the engineering design process), uncertainty quantification and management, and complex system design. Areas of technical expertise include uncertainty propagation methodologies, Bayesian statistics and modeling, stochastic consumer choice modeling, optimization, and design automation. Prior to returning to academe, Dr. Hoyle spent 15 years in industry as a project engineer

and engineering manager, concerned primarily with electronics packaging and with managing the trade-offs between performance, manufacturability, and cost.

Dimitra Giannakopoulou Dr. Giannakopoulou is a Research Computer Scientist with the NASA Ames Research Center, and a member of the Robust Software Engineering Group. Her work is concerned with applying modular and compositional formal verification techniques to autonomous systems and architectures. Before joining Ames, she was a Research Associate with the Department of Computing, Imperial College, University of London, UK, working on methods for the specification and automatic verification of distributed systems. She has graduated from the Dept of Computer Engineering and Informatics, University of Patras, Greece. She hold an MSc with distinction from Imperial College, in "Foundations of Advanced Information Technology", and since March 1999, a PhD degree from Imperial College, University of London.

Guillaume Brat Dr. Brat is employed by Carnegie-Mellon University and he conduct research in software verification within the Robust Software Engineering group in the Intelligent Systems Division at NASA Ames. He received an M.Sc. and Ph.D. from the ECE Department at The University of Texas at Austin. He is Principal Systems Scientist at CMU Silicon Valley serving as an IPA at NASA Ames Research Center. He has been the Assistant Area lead for Robust Software Engineering since October 2009. The group conducts research on new verification and validation techniques, mostly based on formal methods.