AN ABSTRACT OF THE THESIS OF

Scott Kramer for the degree of Master of Science in Mechanical Engineering presented on May 21, 2009.
Title: Extension of Early Stage Failure Analysis Tools to Prognostics Health Management Design.

Abstract Approved:

Irem Y. Tumer

Abstract.

Modern complex mechanical systems have evolved to a point of incredible complexity. Many now operate with significant assistance from automated response functions, and some operate without human operators at all. These increasingly complex systems have also become more expensive and more ambitions in mission, bringing about higher risk to both money and life. In response to these risks, Prognostics and Health Management (PHM) systems are developed to detect, manage, notify, mitigate, and respond to the potential failures in these high risk complex systems. The responsibilities carried by PHM systems make them as integral to the successful operation of a complex machine as any other major system. While methods have been developed and are in use to consider risk and reliability in the early stage of complex system design, explicit consideration of PHM capability and architecture are often left for later stages in the design process. In this work, several risk and reliability based design techniques are discussed with consideration of PHM development. In particular, the Function Failure Identification Propagation

(FFIP) framework provides a systematic process to identify potential failure points and the resulting functional losses from a model based architecture. Research into PHM and embedded system co-design indicates that FFIP is a good starting point for early stage PHM architecture development. In this body, FFIP is first augmented to more completely capture the data needed for preliminary PHM architecture and to begin to address PHM capability in early stage complex system design. Second, an embedded systems design process centered around StateCharts is developed for specific application to the FFIP process in order to facilitate PHM system design during early stage complex system design.

Extension of Early Stage Failure Analysis Tools to Prognostics Health Management Design

by

Scott Kramer

A THESIS

submitted to

Oregon State University

In partial fulfillment of

the requirements for the

degree of

Master of Science

Presented May 21, 2009

Commencement June 2009

Master of Science thesis of Scott Kramer presented on May 21, 2009.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Scott Kramer, Author

ACKNOWLEDGEMENTS

The author expresses sincere appreciation to:

AFSOR Grant BAA 2007-1 : A Framework for Designing Reliable Software-Intensive Systems

United States Coast Guard

Dr. Irem Y. Tumer: Major Advisor

Dr. Toni Doolen: Committee Member

Dr. Lech Muszynski: Committee Member

Dr. Robert Paasch: Committee Member

The friends I’ve made during my OSU academic endeavors

Mom & Dad

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

CHAPTER 1

GENERAL INTRODUCTION

The work contained herein this thesis is completed with the support of AFSOR grant BAA 2007-1 titled A Framework for Designing Reliable Software-Intensive Systems. Quoting directly from the proposal “the overall goal of this research is do develop a formal framework to be used during conceptual design for identifying potential functional failures of complex software-intensive systems, their propagation through the many subsystems that make up such systems, and the resulting system-level impact.”

The challenges of this goal are broken down into five specific elements. Once again, quoting the proposal directly, the objectives are:

(1) to enable the modeling of these interactions qualitatively, consistent with the limited, incomplete knowledge available during the conceptual phase of design.
(2) to enable the identification of potential failures and their propagation paths that are computable directly from the system’s functional and structural topology, their mapping, component interactions, and the resulting dynamics.
(3) to develop a proper mapping between the behavior of the system, its state, and the system functions to enable the identification of non-trivial, non-linear fault propagation paths.
(4) to identify such functional failures that are not directly related to component failures, but still result in a potential loss of functionality.
(5) to enable system designers to include software functionality, failures, and requirements in the initial system design trades for physical systems.

The attached conference papers address the main objectives and the specific challenges listed here through progressing the existing Function Failure Identification and Propagation (FFIP) analysis framework from a tool that reasons failure propagations through analysis of the functional relationships between the mechanical components of a complex system to an early stage software model that captures the system hierarchy in a logically clear computational model. As FFIP is a failure analysis method, the extension here is directed toward making it a Prognostics and Health Management (PHM) design tool. PHM refers to be the embedded automatic response system that includes the sensors, processors, computational hardware, and code to detect, notify, and often automatically respond to failures within a complex system. Because PHM

carries the responsibilities of detecting, notifying of, and responding to complex system failures, it is appropriate to consider PHM design around a failure analysis methodology. This work provides a novel process to begin the critical step of PHM architecture design during the conceptual stage of complex system design in a manner a that allows engineers to consider PHM capability early on based on preliminary system simulations. With PHM development started in a meaningful way early in the design process, PHM architecture can grow with overall system refinement allowing for more opportunity for concurrent consideration, design option exploration, and system testing during the PHM design process.

The first paper, titled "A Framework for Early Assessment of Functional Failures to Aid in PHM Design"

lays out the PHM development problem through research into what is needed in order to develop a preliminary PHM model during early stage complex system design. It is determined that intelligent PHM design requires a qualitative understanding of component interactions and understanding of failure propagation paths throughout the system. A qualitative understanding of the system allows PHM designers to recognize functional failures, and an understanding of failure propagation paths allows PHM designers to address automatic failure response programming. Several failure analysis methodologies are researched and FFIP is determined to be well suited as a starting point for early system design PHM architecture development, although other failure analysis methods are still considered of significant value for informed design. FFIP, even without modification, addressed grant objectives 1 and 2 well, but in order to consider important, but still very general principles that hereto have been omitted, it is modified in the first paper in three significant ways. First, non-linear failure propagation is added to existing configurational diagrams. This captures the reality that some component failures are catastrophic enough to damage other components in physical proximity, yet outside the functional flow of the failed component. Second, a time to failure propagation estimate is added to the graphical framework of FFIP. Although the finite details of failure propagation are not known at early stage system

design, order of magnitude estimates are still critical for PHM architecture. In short, failures that propagate slowly may be addressed through PHM response, while failures that propagate very quickly through the system must be addressed in some other way. From an early stage, system engineers can begin to understand and incorporate fundamental PHM capabilities in the system architecture. Finally, question of how PHM response affects the overall system architecture is addressed. The Function Failure Logic (FFL) reasoner is modified to explicitly embrace how an embedded PHM system would respond to modeled failures. This becomes important when a PHM response changes a component state to respond to a failure, that component state change may have consequences that need to be considered.

All of these modifications are applied to a liquid fuel rocket engine model of moderate complexity. This model was chosen to exercise the modified FFIP architecture beyond the conceptual examples already in the literature, and to adhere to the theme of the proposal as applied to general liquid rocket architectures.

The second paper titled “Application of the StateCharts and FFIP Methodologies to PHM Architecture Co-Design” addresses the remaining objectives through synthesizing FFIP with embedded system design principles. In this paper, a brief review of the most common embedded system design languages is discussed along with strengths and weaknesses as far as application to early stage PHM design is concerned. As this is to be an early stage conceptual design tool, early design stage embedded system design processes were researched including Petri nets, Specification and Description Language (SDL), and StateCharts. Each process has strengths and weaknesses. Petri nets contains explicit logic capabilities that aren’t easily represented through SDL or StateCharts, but quickly looses modular clarity with increasing complexity. SDL retains the modular nature of the finite state reasoning used in FFIP, but focuses on capabilities more appropriate for refined system design. The StateCharts language is selected and applied as appropriate to the FFIP nomenclature. The StateCharts representation of the system fulfills objective 3 in that the system is decomposed into hierarchies ranging from the high level system down through functions and sub functions eventually to components. Components map to the StateCharts hierarchy in a

modular nature, just as they map from configuration flow graphs to functional flow graphs. In StateCharts, however, functions are explicitly stated as opposed to hidden along functional flows in a configuration diagram. Finite state logic is applied to these StateChart hierarchies that develops the process to build a more robust FFL reasoner. This FFL reasoner is essentially a very early stage computational model of the complex system. Each system function is listed in a StateChart that is linked to levels of the system above and below that function. The StateChart contains the programming logic for the state of that function that will be determined by command inputs, component states, and sensor readings. This logical capability, that includes inputs beyond component states, satisfies objective 4 in making functions dependent on the more complex realities that can be recognized by designers as the system is broken down into this hierarchical StateCharts model.

Once the complex system is translated into this StateCharts language and a FFL reasoner is constructed with the StateCharts finite state logic, the process of PHM development is discussed in detail. The culmination of improving FFIP and then applying state charts to it, is where objective 5 is addressed. Whereas the StateCharts constructed thus far represent the system and are used to simulate system behavior, the PHM StateCharts constructed next represent the programming of the embedded PHM system itself. Through the use of the FFL reasoner and additional considerations from other risk and reliability analysis tools, engineers design PHM responses to specific failures. The StateCharts model of the system is used to develop the sensor fingerprint for specific failures that feed an adjacent logic structure that contains the appropriate responses to failures that require an automated response. Those responses then feed back into the StateCharts model of the system in the form of command signals at the appropriate points in the hierarchy, effectively simulating the system performance with an active PHM. Further, the PHM StateCharts check for response effectiveness, the first step in addressing software failure scenarios through recognition of ineffective responses. This, ultimately, is the consideration of a complex mechanical system that is governed by automated software response through various operational and failed states during early stage system design. As in the first paper, the methodology is applied to the same liquid fueled rocket engine architecture.

Future work is also discussed in both papers, but the future work in the second paper addresses the need to explore deeper into software reliability analysis application. Although FFIP is improved and the state charts process already begins to address software reliability, there is significant room for improvement. As is the case with risk and reliability analysis tools for mechanical systems, the tools for software systems can only find failure modes, not confirm the absence of them. As this is the case, future work should concentrate on finding or creating the best software risk and failure analysis that can be applied to state charts in early stage system design, and using them to further the usefulness of this methodology.

CHAPTER 2

A FRAMEWORK FOR EARLY ASSESSMENT OF FUNCTIONAL FAILURES TO AID IN PHM DESIGN

Scott Kramer

Irem Tumer

ABSTRACT

Human artifacts have evolved into incredibly complex systems that rely on both hardware and software to function dependably with limited or no human operator control. As the monetary and human consequences for failure climb with technological progress, Prognostic and Health Management (PHM) systems are developed to help manage, mitigate, detect, and respond to failures. These PHM systems have become as integral and as important as any other subsystem in a complex machine. Methods have been developed to analyze and incorporate risk and reliability in the early stage decision making processes of complex system design. These methods, however, treat the development and capabilities of PHM in the conceptual stage of system design in a peripheral manner at best. The importance of PHM consideration early in system design is significant. The structure of PHM, one of the most complex of the subsystems, can begin to take shape. Design modifications that must be made in light of PHM limitations should be pushed to the earliest stage of design possible, where costs of changes are minimized. In this paper, several risk and reliability based design techniques are discussed in this context. In particular Function Failure Identification Propagation framework (FFIP) provides a systematic process to identify potential failure points and their resulting functional losses. FFIP is selected in this paper as fitting early stage PHM development very well. However, improvements are proposed to the existing FFIP process to better address PHM design needs during the conceptual design stage of a complex system. This paper presents an improved method, which is then applied to a liquid fueled rocket engine architecture. Future work, including using the information gathered here to start the conceptual design of the PHM system, is also discussed.

INTRODUCTION

Prognostics and health management (PHM) systems are integrated and embedded hardware-software systems that address the management of the 'health' of highly complex engineered systems. PHM systems are charged with determining the condition of each element of a complex system and providing data, information, and knowledge (DIaK) to control systems and operators. These systems must evaluate the condition of system elements, detect anomalies, diagnose causes, determine overall system state, predict system impacts, communicate DIaK in a timely and contextually accurate manner, and initiate automatic responses in the control programming. This is a tall order, and in order to develop this system architecture, PHM engineers sometimes must wait until significant detail is known about the complete system design. Even then, PHM is usually a conglomerate of fault detection and isolation of variant subsystems instead of an integrated system-wide health monitoring application. Some of this can be blamed on the evolution of human artifacts, as the first PHM came into being with the first sensors that provided information to operators. As systems became more complex, some of these sensors were integrated into command and control systems and would initiate automatic responses to prescribed sensor readings [1]. Today, PHM is required to be an integral part of controlling and operating complex systems that are too complicated to rely on human operators to be responsible for all system functions, or operate without human operators at all [2]. Fundamentally, PHM must be able to answer key questions about the state of a system that were the responsibility of human operators for as long as complex systems have been around including: "What are the current system capabilities? How safe is the current state condition? How can the current degraded elements and failures be remedied or contained?" As PHM has become absolutely critical to complex system operation, it is vital to treat PHM design as an integral part of the complex system from the earliest possible stages of design [3].

PHM development must rely heavily upon system modeling, about which much has been developed for the advanced stages of the design of complex systems. Although still a field of very active research, serious PHM development is often not addressed

until very detailed physics based modeling is developed [2]. There is significant work done on developing algorithms, programs, and sensor technologies leveraged toward PHM development, but nearly all of these require a refined design state and focus on a small subsystem of the larger machine. While these are all valuable for the later stages of design, the methodologies developed for PHM architecture development early in the design stage are limited. It is often the case that modern PHM systems are conglomerates of subsystem PHM development that fail to consider interactions between subsystems while capturing interactions within subsystems [4]. Pushing the start of PHM development to the conceptual stages of complex system design also addresses the critical issues of exploring how PHM itself changes the states of complex systems. An early start on PHM also begins to explore the limits of PHM in terms of what problems PHM can fix while in operation, and what problems propagate with too much severity at too fast of a pace for PHM to effectively address.

A critical component of modern complex system design are the application of risk and reliability analysis techniques. Some of these methods have been used in practice and are commercially used as starting points for PHM design. Some methodologies can be used in the conceptual design stages, and many of them ask the same questions of engineers that must be answered by the eventual PHM system.

In the following we first summarize the most common risk and reliability tools used in complex system design and a brief analysis of their application to PHM design during conceptual system design. We then focus on the Function Failure Identification and Propagation (FFIP) method developed by Tumer and Kurtoglu [5] which provides a very solid foundation for the development of PHM systems during early stage design. The FFIP process depends on three major modules: 1) a configurational model that contains the components of the system; 2) a functional model that contains the functions of the components of the system, and; 3) a function failure reasoner that facilitates a high level behavioral simulation of nominal and failed states for a system in the early stages of conceptual design. This process begins to address the fundamental questions that PHM must address from an early stage, but as this process

has not explicitly been used for PHM development, modifications and improvements to the FFIP process are then proposed to better enable PHM development. In this paper, FFIP is briefly described using an application to a moderately complex liquid fuel rocket engine as an example. The liquid fuel rocket engine is chosen as it is a good example of a complex system that operates with minimal or no direct human control and has significant failure consequences. Then, three augmentations to FFIP are introduced to better suit it for PHM development. Finally, future work is discussed including a further PHM development process that builds this work into a state charts based PHM architecture.

BACKGROUND

Risk & Reliability Analysis: Assessing risk and reliability during design can be done using tools such as FMECA, FFA, FFIP, PRA, and many others, which utilize modeling techniques as a critical element of capturing and quantifying system risk. Failure modes effects analysis (FMEA) or Failure modes effects and criticality analysis (FMECA) is a process which examines each component for their failure modes, the risk that those modes can have, and the criticality that can result from those failures. FMECA is currently used in industry to gather information for the development of PHM systems, and can augment the methodology proposed here [6]. Fault tree analysis (FTA) aims to capture paths of failure events from root causes up to ultimate consequences [7]. Although both FMECA and FTA can be invaluable in the understanding of the risk in a system design, neither of them are model based, and as such, they cannot form a solid foundation for the development of a PHM architecture.

Finally, Probabilistic Risk Assessment (PRA) combines the use of several techniques involving fault/event modeling techniques and uses a probabilistic framework to guide decision making [8,9]. The problems with using PRA in early PHM design are 1) the probabilistic nature of components is not qualitatively known during early design stages; and, 2) the process is not model based, an important characteristic for the development of PHM architecture.

Model Based Diagnosis

Diagnostic reasoning approaches rely on a process where a system is monitored by a group of sensors and the readings of those sensors are compared to the expected behavior of the system to detect unusual conditions. Specifically Model Based Diagnostics (MBD) and other related techniques are important to both PHM and system design [10,11,12,13,14,15,16]. MBD, however, relies on a system design that

has advanced beyond the early stage and does not allow PHM architecture design to begin in the early stage of system design.

The FFIP process, developed by Tumer and Kurtoglu [5] combines early-design modeling with model based diagnostic reasoning. Specifically, FFIP is a model based risk analysis tool which was developed to be applied to the early conceptual stage of a complex system design. In this process, failure modes of components are identified and simple finite state models are developed to predict the system impact of a failure and the overall system state. Because FFIP uses the actual conceptual architecture of the complex system and starts to document the simulation process, FFIP is potentially a good starting point for the development of PHM architecture. However, since FFIP does not quite satisfy the needs of PHM development, FFIP is modified in this paper to address the fundamental needs of PHM design.

FFIP

The FFIP framework combines the modeling of function, structure, and behavior to simulate failure propagation paths and the ultimate functional failures. This information can be used to determine mitigation options in the design stages of complex systems. Currently, the three major modules in FFIP analysis are the graphical system model, the behavioral simulation, and the function-failure logic (FFL) reasoner [17]. The system is modeled using graphical representations. Function models are used to represent system function in FFIP [18], which are representations of a system that shows the overall system functions decomposed into smaller, more fundamental sub functions. The subfunctional relations are shown through the flows of energy, material, and signal (EMS). The Functional Basis taxonomy is the standardization used for the construction of these models. The structure of the design is captured in a configuration flow graph (CFG) [19]. Linked to the functional model above, CFGs strictly comply to the functional topology of the system [20]. Each node

of the CFG corresponds to a system component, and arcs represent energy, material, or signal flows using the same terminology as the Functional basis.

Finally, the behavior of the system is modeled with a component oriented approach. High-level, qualitative behavioral models of components are developed in various nominal as well as faulty modes. The behavior of each component is derived from input-output relations and underlying fundamental principles. The finite states of each component are defined while the physical laws and mathematical details are defined separately. These physics based modeling details are left for later in the design process. There are many processes for modeling system behavior early in system design, but most model only nominal behavior. FFIP focuses on modeling failed states of components and monitoring the relationship between behavioral dynamics and component function.

The simulation process is set to determine the system behavior under certain conditions. Conditions here represent event occurrences that cause component mode transitions. Throughout the simulation, both the component modes and the set of system state variables are tracked. As such, the overall system state at any given time is a function of the component modes and the system state variables:

$$X(t)=F(c(t),v(t)) \qquad (1)$$

where $c(t)=[c1, c2, c3\ldots,cN]$ is a vector of discrete component modes and $v(t)=[v1,v2,v3,\ldots,vk]$ is a vector of system state variables.

As the quantitative details of the system cannot yet be determined at this early stage, finite states of each element are assigned instead. For example, liquid flow rates for

valves can be assigned a set of {zero, low, nominal, high} and a sensor signal may have values of {no signal, high, nominal, low, zero}. With each component simplified to a finite state element, laws can be built governing these components, treating them as portable modules that can then be fit together to build a functional model of the system [5]. This is exemplified in Figure 2.1, borrowed from [5].

The 'functional' nature of FFIP satisfies most of the needs of a PHM specification. This process defines system states and conditions, predicts system impacts of failures, provides some of the information needed to design the detection apparatus and write the diagnostics logic. Missing from FFIP are temporal data and a complete failure mitigation and response reasoning. FMECA can address the response reasoning, but temporal data regarding time to failure and time to propagate from any initial failure must be added to the FFIP analysis, if only on an 'order of magnitude' basis. It is worth noting that such temporal information may be little more than 'best guesses' for much of the design process. Regardless, 'order of magnitude' data can be quite useful in the arrangement of the embedded system architecture.

APPLICATION: PHM Design for Liquid Fueled Rocket Engine

The existing FFIP framework is briefly explained using a moderately complex Liquid Fuel Rocket Engine (LFRE) architecture as a working example. The model presented here is a basic staged combustion model similar to those used in the Space Shuttle operated by NASA. In this model, liquid hydrogen is used as both the fuel as well as the coolant. The first pressure boost the hydrogen flow receives comes in the low pressure fuel booster, which is driven by hot gasified hydrogen carrying heat energy from the regeneratively cooled tubular nozzle. A coolant control valve regulates the amount of hydrogen that flows through the regeneratively cooled nozzle sending the hydrogen that is not used for cooling directly to the preburners. The hydrogen is combusted in preburners with some of the oxygen that drives turbopumps on both the fuel and oxidizer flows. The low pressure oxygen turbopump is driven by liquid oxygen pressurized by the high pressure oxygen turbopump that is connected to the oxygen side preburner. Some of the oxygen is burned in the preburners and the remainder is injected into the main combustion chamber where it burns with the hot gasses generated in the preburners [21].

Applying FFIP to this LFRE architecture: The configurational flow graph (fig. 2) lists the components and the flows between them. The configurational flow graph maps directly to the functional flow graph (fig. 3) where the functions of the system are clearly laid out. The flows delineate the transfer of signal, energy, and material between the different functions and components. In prior FFIP applications in the literature, the flows of signal, energy, and material are considered separately, but in this example, some of the hydrogen flow and some of the oxygen flow is represented as both energy and material in the same flow as energy is carried back to the low pressure turbopumps by these material flows.

Using the finite state logic developed earlier (Fig. 1), a Function Failure Logic (FFL) reasoner is developed that reflects this LFRE system. Although the finite details about the operating parameters of this system have not been developed, a simplified early stage simulation tool is developed that will show how the failures of components within the system will propagate through the system and potentially result in a change in the overall system state.

As it stands, FFIP provides much in the way of information needed to develop the PHM architecture. A finite state model of the system is developed early on, and important relationships regarding potential responses to component failures are developed. The use of FMECA and other risk analysis techniques add to the knowledge base needed for PHM development. Regardless, certain critical needs of PHM development are not currently addressed with FFIP.

AUGMENTATIONS TO THE FFIP METHODOLOGY

To improve the early design stage collection of data, information, and knowledge needed for developing PHM systems, some critically important additions to FFIP are proposed in this paper. First is a time to failure reasoning, and second is a proximity propagation logic, and third is an operational response mitigation paradigm. These steps are critical to complete the data, information, and knowledge set needed for the PHM architecture.

Augmentation 1: Time of propagation analysis.

Critical to the effective response of a PHM architecture is timeliness. This is fundamentally important for not only the early development of PHM architectures, but also to serve as an early design stage indicator to the rest of the engineering team as to just what the limits of PHM are with regard to component failure response. We are concerned about three catastrophic system failures that originate somewhere in the system and propagate through: "1) failures that propagate in a manner that a PHM architecture does not have the opportunity to provide adequate response for; 2) catastrophic system failures that propagate quickly and hence provide very little (but some) opportunity for response and mitigation, and 3) failures that propagate slowly and provide ample opportunity for response and mitigation. These are represented on the diagrams with a "TTFP" label indicating the estimated time to failure propagation. Currently FFIP considers a 'time step' architecture that tracks how variables change throughout system operation. From this, a nomenclature describing the temporal consequences of a degraded or failed state can be inserted into the architecture clarifying the speed at which consequences 'propagate' through a failure path.

Although the overall system architecture may not provide ample data for high level timing detail, the process of grouping failures into these categories is an important step

for the design of the embedded reactive PHM system. Once again, a simple finite state logic can be used to indicate how quickly the change of states flows through the system architecture.

Table 1 illustrates the first scenario in which the failed component is the coolant regulator valve. In this LFRE architecture, the fuel, hydrogen, also acts as a coolant for the main nozzle. The heat that the hydrogen absorbs as a coolant is used to power the low pressure fuel booster that draws the hydrogen from upstream in the system. At time step one, the coolant control valve fails in the closed state (due to clogging, or malfunction). This results in no hydrogen bypassing the nozzle heat exchanger and all the hydrogen coming from the fuel turbopump going through the nozzle heat exchanger. As a result, more thermal energy is delivered to the low pressure turbopump thus increasing the overall flow of hydrogen to the system over soe period of time. This is modeled through the function-failure reasoner in Table 1. Although impossible to model with finite accuracy at an early design stage, indications are that the system would stabilize at this slightly higher flow of hydrogen, and, assuming nominal flow rates, would not result in catastrophic component failure immediately. Given enough time, however, the system will expend fuel at a rate faster than nominally, and changes in vibrations could eventually result in catastrophic failure. Although specific timing details are not known, it can be shown that a failure of the coolant control valve would have a moderate propagation time with potentially extreme consequences. Here we can indicate that a failed coolant control valve is a ‘time type 3’ failure where a PHM reaction that does not result in entire system loss may be a reasonable mitigation strategy. This nomenclature is added to the FFG in Figure 2.4 by the “TTFP:3” nomenclature as well as to the CFG. For clarification, a “TTFP: 1” failure would be a loss of oxidizer containment that may result in an explosion, where a “TTFP:2” failure would be a loss of preburner function where PHM could be used to respond to the casualty by shutting the system down, hereby preventing the worst case scenario (possible explosion) at the sacrifice of system

function (loss of propulsion). Note that these groupings only speak to PHM ability to respond with regard to how fast a failure moves through a system and not the ability of PHM to respond to a particular type of failure.

Examples of a failure that PHM cannot reasonably expect to be able to respond to is a leak of oxidizer. In the event of massive oxidizer leak, a violent explosion becomes possible and a large fire is likely in a very short time. PHM may be an important part of responding to such a failure, but it is not reasonably expected that PHM can handle this alone. As such, engineers should look elsewhere to prevent this failure mode.

As an early analysis of the speed at which these failures propagate can start to clarify the limitations of PHM early in the design, changes to the overall system design that can avoid these limitations can be made much earlier and at significant cost reductions using this augmentation.

Augmentation 2: Proximity propagation between flows.

Another practical realworld consideration not clearly addressed in FFIP is how failures may propagate laterally through a complex system [22]. For example, a pipe failure can lead to fuel or oxidizer entering a compartment that requires a coolant pump for the central computer cooling system to be shut down. FFIP addresses the configuration of a complex system and looking at this lateral failure propagation is a logical addition to the functional failure analysis of the overall system. From a PHM system design perspective, these lateral failures may require reactions in other subsystems and may reveal additional sensor resources needed for the failure verification process, an issue of critical concern for PHM reliability, if specifically addressed in this work.

The configurational flow graph (CFG) model can be used to capture some of these relationships. Because the model takes into account a relative proximity, designers can take note of potentially important proximity relationships from an early stage. As it stands, the functional model and configurational flow graph track the flow of signal, electrical, and physical flows through the system along the functionally designed paths. Failures or changes of state that occur are modeled within these paths and propagated down stream. In reality, certain failures can propagate across these streams. An example of this is a component explosion that can send debris, hot plasma, electrical discharge, flammable liquid spray, or other hazards across the physical space of a complex system and interact with a completely different flow. In a scenario this extreme, it seems unlikely to avoid a catastrophic failure of a system were this to occur, making the analysis of such failures critical to the early design process and the principles of damage control in complex systems. PHM can play a significant role in damage control. If PHM can recognize such a component explosion and the proximity of other vital subsystems are considered in the automated response, PHM can automatically shut down other systems, re-route critical flows, or even shed components from the system. The LFRE model used up to this point is not the most optimal opportunity to exemplify this process, but it can be shown by moving 'up stream' from the engine module along the liquid oxygen supply. At some point, it is likely in any design involving the transport of potentially volatile liquids that there will be proximity to other vital flows. In the example here, the liquid oxidizer flows near a control signal bundle. In the event of a catastrophic isolation valve failure and the transfer of energy laterally across flows, the control signal bundle could be cut, depriving the system of vital sensor data and component control.

Here, we added a lateral propagation nomenclature to the configurational flow graph and functional flow graph nomenclature indicating that a lateral failure of the fuel valve is high (LP:3) while the lateral failure of the control conduit box is low (LP:1) shown in figure 2.5. Because we have a lateral failure path leading from the fuel valve

to the conduit box, we have drawn an arrow indicating a new flow of energy from the fuel valve's lateral propagation tab to the conduit box. Since the failure propagation going the other direction is not deemed possible, no arrow is drawn. From this, we have made an observation early in the design stage that allows engineers to consider the lateral failure and begin to consider mitigation very early on. From a PHM perspective, a smart PHM system that is programmed to understand these lateral failures has another method to use in the identification and assessment of specific component failures. Building on this, a separate Configurational Failure Logic (CFL) reasoner can be built that addresses these lateral proximity failures. This is difficult to build into the existing FFL reasoner in FFIP, because the existing finite state machine works to simulate the model in failed states where failures will propagate along existing functional paths. Here, the failure propagation analysis depends on more than the simplified finite state models used for analysis up to this point. This analysis becomes somewhat more nuanced as the relationships are not definite, but are instead probable. Shown in figure 2.6, The logic of the proximity based CFL is similar in scope and content to the FFL logic, but instead of a focus on function, the proximity based CFL focuses solely on the components themselves. Inputs to the proximity based CFL come from the FFL reasoner as that, through the functions of each component, will determine the operational status of each component.

This logic can be used to build a larger spreadsheet similar in format and content to the FFL that can demonstrate the potential lateral propagations of these failures. It is worth noting that this is somewhat of a departure from earlier noted advantages of FFIP in that there is more 'judgment, experience, and subjectivity' needed here. Also worth noting is that other early stage failure analysis processes such as FMECA and PRA address this type of scenario. The advantages presented by bringing this to FFIP are that this process is incorporated into the actual conceptual layout of the system, making proximity propagation easier to see, easier to communicate to the entire design team as they are visually represented as potential flows in a system. As a consequence,

this augmentation allows an early look at this vital issue before going so far down the design process that actual physics based simulation is being completed to capture these failure modes. Another interesting example worth nothing where such a scenario could be encountered would be if the PHM subsystem of the challenger were able to detect a fatal seal leak and separate the orbiter from the booster stack before the infamous ignition. These non-linear yet interdependent relationships can be very complex and many may not be known until the design has reached a higher level of refinement. An example of this would be a 'heat sharing' scenario where a small compartment on a spacecraft holds three elements of three separate function flows, yet each relies on the latent heat created by the other elements for optimal operation. The configurational and functional architecture of FFIP can make such nuanced relationships apparent to attentive engineers earlier in the process, and this can become critical to the data development of the PHM system.

Augmentation 3: Explicit statements of PHM response.

The final enhancement to FFIP proposed here is the addition of response simulation. If any given failure falls into a category that can be addressed with a PHM response, the desired response needs to be identified early on. Often, since the PHM response will require the modification of the state of another component or components, the process can look directly at the results of those modifications essentially considering them as new 'mitigating' failures that will have their own important repercussions worth considering. This further builds the early stage finite state machine that is critical to the PHM software architecture.

The proposed method for accomplishing this is an increase in the complexity of the FFL reasoner. As it stands, the FFL models the system response to a discrete failure or change of state as it propagates through the preliminary system architecture. Adding a column to the right of the simulation tool that contains the logic for failure

response that a PHM would be programmed to take. As FMECA is a brainstorming tool that uses experience of the design team to select appropriate responses a PHM may take to given failures, similar brainstorming and use of experience concepts can be used here. In this new column, a finite state logic will institute appropriate modifications to the FFL automatically. This more complex simulation will allow engineers to begin to vet out how the complex system will react with the inclusion of a PHM architecture. Essentially, this is a form of building the software logic of the PHM subsystem in conjunction with the system architecture itself.

Using the first failure example applied to our LFRE model, our current function-failure reasoner shows that the failure of the coolant control valve to state {clogged} can result in an increased flow of fuel through the system as more energy is carried to the low pressure fuel booster turbo pump. As this could result in increased fuel flow that will bring the fuel oxidizer ratios out of balance and increase the fuel usage rate, the PHM system can be programmed to verify this expected response using the appropriate sensors and then throttle down the fuel inlet valve from {nominal} to {low}, effectively bringing the overall fuel flow rate back down as demonstrated in the FFL reasoner in Table 2.

Although it is probably very apparent to the reader that building more complexity into the FFIP process through the addition of responsive logic will make the finite state simulation more tenuous and more difficult to reason, there is still significant value in starting to build the fundamental architecture of the PHM system. Using this to build the response programming of the PHM system during the conceptual design stage allows engineers to incorporate this response knowledge into the more detailed physics based models that will follow as more refined designs are approached.

CONCLUSIONS

In this work, the Function Failure Identification and Propagation (FFIP) process is built upon in order to build it into better meeting the needs of the Prognostic Health Management (PHM) design process. Driven by the ever more dangerous missions and systems that incur increasing costs, the design of complex systems has gotten to the point that recognizing system weaknesses, failure modes, and mitigation strategies early on is increasingly important. Driven by the increasing complexity, human operators are quickly receding from the forefront of command and control. As systems out pace human ability to respond to failures or changes of state, the integrated prognostics and health management systems become ever more important. PHM is now an integral part of any complex system design. As a result, PHM is becoming the main manager of operational system risk. This paper investigates how FFIP, a model based failure analysis technique based on the functional and configurational framework of early system design can be leveraged to impact the development of PHM.

Building on existing novel advantages of FFIP, the framework presented in this paper increases the ability of engineers to garner the data, information, and knowledge resources of the design team and leverage them toward the design of PHM architecture from the very start of the conceptual design process. The existing FFIP architecture is augmented through the addition of a time to failure propagation nomenclature, a consideration of lateral propagation across subsystems that may not have shared functional flow paths, and through an explicit consideration of PHM response in early stage system simulation.

This process not only shows how failures propagate through a system along established flow paths, but also starts to investigate how quickly these failures propagate and analyze the ability of a PHM system to address any given failure from the very early design stages. Failures that can propagate across flows through energy or material transfers that violate the established flow routes are also explored and a formal framework for their early stage analysis is proposed. Finally, this process

explicitly looks at how PHM simulation can be used in FFIP to begin to see how a PHM system may act upon a conceptual design of a complex system.

FUTURE WORK

The work described in this paper represents critical additions to the FFIP architecture in its application to PHM development. Some of the limitations of PHM's scope of responsibility have been explored. To further this work, the processes of embedded system design must be applied to the FFIP architecture. Based on this augmented FFIP, a process will be developed that creates the conceptual outline of the PHM architecture itself. Based on a state charts type methodology, a common framework for early stage hardware/software architecture development [23], the continuing work will create a process that allows hardware and software engineers to develop the programming that makes PHM possible. This state charts type architecture will allow more explicit consideration of sensor selection, placement, reliability, and heuristic capability and limitations of the automated nature of a PHM subsystem from much earlier in the design process, better allowing the inclusion of PHM development into the fundamental system design from the conceptual design stage [24]. Also, a more explicit consideration of the software side of the PHM system will allow application of software specific reliability considerations from the earliest stages of overall system design [25,26,27,28,29,30]. The augmented FFIP framework discussed in this paper can be further improved through implementing and including standardized software development architectures, further exploring the limitations of practical PHM application and beginning the incredibly complex process of developing the coded software package ever earlier in the design process.

TABLES & FIGURES

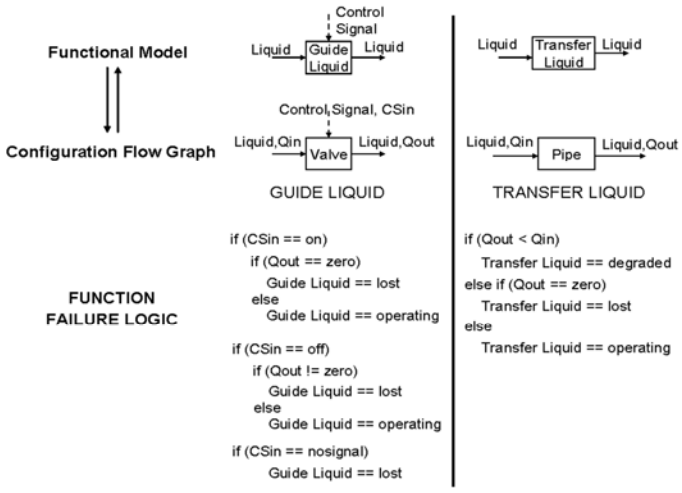

Figure 2.1: Function-Failure logic for generic pipe and valve components [5].

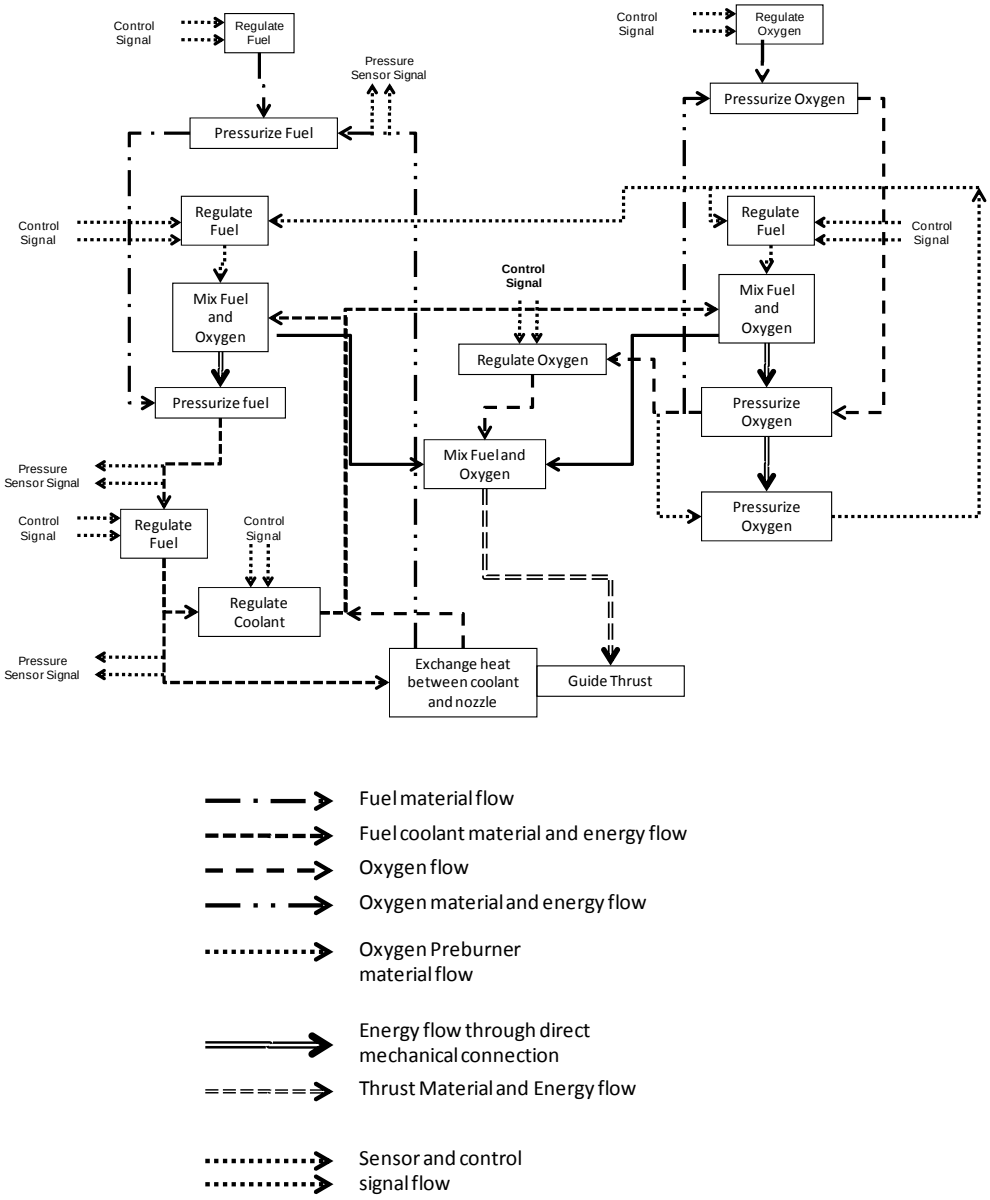

Figure 2.2: Configurational Flow Graph (CFG)

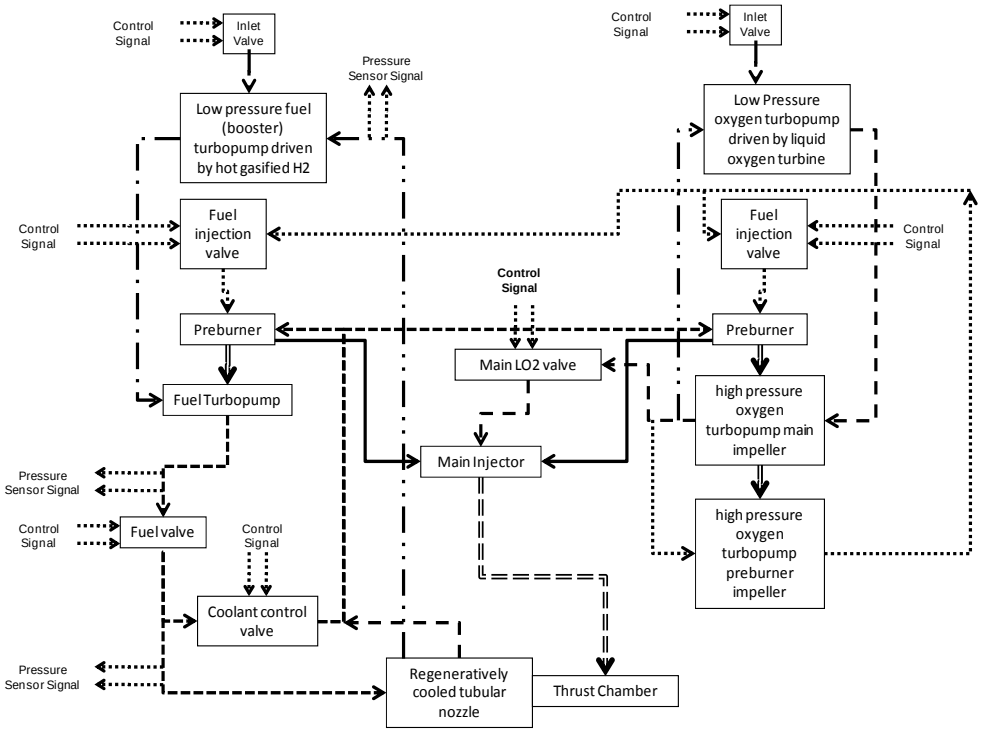

Figure 2.3: Functional Flow Model

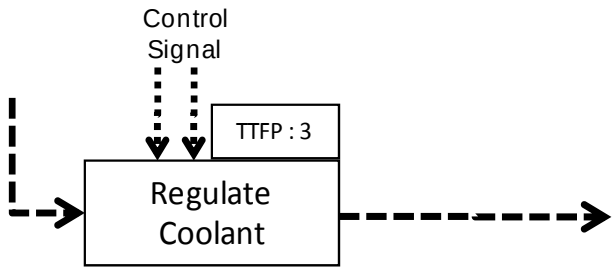

Figure 2.4: Time to Failure Propagation consideration added to FFIP graphical representations.

Table 2.1: FFL logic applied to the Liquid Fuel Rocket Engine coolant valve failure.

	at t=0	at t=1	at t=3	at t=4	at t=5
Component Modes					
Inlet Valve	nominal	nominal	nominal	nominal	**high**
Low Pressure Fuel Booster	nominal on	nominal on	nominal on	**high**	
Fuel turbopump	nominal on	nominal on	nominal on	nominal on	**high**
Fuel valve	nominal on	nominal on	nominal on	nominal on	**nominal on**
Fuel valve controller	nominal on	nominal on	nominal on	nominal on	nominal on
Coolant control valve	nominal	**failed off**			
Coolant valve controller	nominal on	nominal on	nominal on	nominal on	nominal on
Regeneratively cooled tubular nozzle	nominal	nominal	**high**		
Preburner	nominal on	nominal on	nominal on	nominal on	nominal on
Low pressure turbopump	nominal on	nominal on	nominal on	nominal on	**high**
Preburner	nominal on	nominal on	nominal on	nominal on	**degraded**
Main combustion Chamber	nominal	nominal	nominal	nominal	**degraded**
sensor	nominal on	nominal on	nominal on	nominal on	nominal on
State Variables					
Liquid Flow	nominal	nominal	nominal	nominal	**high**
Pressure increase	nominal	nominal	nominal	**high**	
Liquid Flow	nominal	nominal	nominal	**high**	
Pressure increase	nominal	nominal	nominal	nominal	nominal
Liquid Flow	nominal	nominal	**zero**		
Pressure control	nominal	**zero**			
Liquid Flow	nominal	nominal	**high**		
Combustion	nominal	nominal	nominal	nominal	**degraded**
Liquid Flow (Pipe 6)	nominal	nominal	nominal	**high**	
Liquid Flow (Pipe 7)	nominal	nominal	nominal	**high**	
Control Signal Flow	nominal	nominal	nominal	nominal	nominal
Status Signal Flow	nominal	nominal	nominal	nominal	nominal
System Functions					
Import Liquid (inlet pipe)		operating	operating	operating	operating
Guide Liquid (LP)		operating	operating	operating	operating
Guide Liquid (Turbopump)		operating	operating	operating	operating
Guide Liquid (Fuel Valve)		operating	operating	operating	operating
Guide Liquid (Coolant control valve)		**lost**			
Process signal (Coolant valve controller)		operating	operating	operating	operating
Guide liquid, heat (regen nozzle)		operating	**high**		
Guide liquid, heat (preburner)		operating	operating		**high**
Guide liquid, heat (LP turbopump)		operating	operating	operating	operating
Guide liquid, (Preburner)		operating	operating	operating	operating
Guide liquid, thrust (Main Combustion Chamber)		operating	operating	operating	operating
Measure flow		operating	operating	operating	operating
Measure vibration		operating	operating	operating	operating

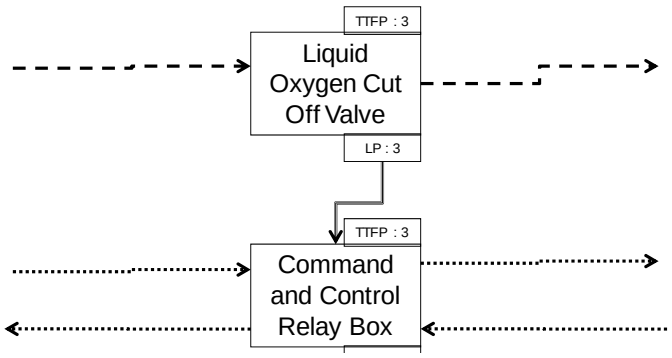

Figure 5: Lateral failure propagation in FFIP

Figure 2.5: Lateral Failure Propagation if FFIP.

```
Command and Control Relay

If (Liquid Oxygen COV == operating)
        Command and Control Relay == operating
else
        Command and control Relay == possibly failed
```

Figure 2.6: CFL Logic for Lateral Failure Propagation.

Table 2.2: FFL Logic with PHM response applied to the Liquid Fuel Rocket Engine coolant valve failure

	at t=0	at t=1	at t=3	at t=4	at t=5	Response
Component Modes						
Inlet Valve	nominal	nominal	**Low**	**Low**	**Low**	
Low Pressure Fuel Booster Flow	nominal on	nominal on	nominal on	nominal on		
Fuel turbopump	nominal on	nominal on	nominal on	nominal on	**high**	
Fuel valve	nominal on	nominal on	nominal on	nominal on	**nominal on**	
Fuel valve controller	nominal on	nominal on	nominal on	nominal on	nominal on	
Coolant control valve	nominal	**failed**				Regulate inlet valve
Coolant valve controller	nominal on	nominal on	nominal on	nominal on	nominal on	
Regeneratively cooled tubular nozzle	nominal	nominal	**high**			
Preburner	nominal on	nominal on	nominal on	nominal on	nominal on	
Low pressure turbopump	nominal on	nominal on	nominal on	nominal on	**high**	
Preburner	nominal on	nominal on	nominal on	nominal on	Nominal on	
Main combustion Chamber	nominal	nominal	nominal	nominal	nominal	
sensor	nominal on	nominal on	nominal on	nominal on	nominal on	
State Variables						
Liquid Flow	nominal	nominal	nominal	nominal	**high**	
Pressure increase	nominal	nominal	nominal	**high**		
Liquid Flow	nominal	nominal	nominal	**high**		
Pressure increase	nominal	nominal	nominal	nominal	nominal	
Liquid Flow	nominal	nominal	**zero**			
Pressure control	nominal	**zero**				
Liquid Flow	nominal	nominal	**high**			
Combustion	nominal	nominal	nominal	nominal	**degraded**	
Liquid Flow (Pipe 6)	nominal	nominal	nominal	**high**		
Liquid Flow (Pipe 7)	nominal	nominal	nominal	**high**		
Control Signal Flow	nominal	nominal	nominal	nominal	nominal	
Status Signal Flow	nominal	nominal	nominal	nominal	nominal	
System Functions						
Import Liquid (inlet pipe)		operating	operating	operating	operating	
Guide Liquid (LP)		operating	operating	operating	operating	
Guide Liquid (Turbopump)		operating	operating	operating	operating	
Guide Liquid (Fuel Valve)		operating	operating	operating	operating	
Guide Liquid (Coolant control valve)		**lost**				
Process signal (Coolant valve controller)		operating	operating	operating	operating	
Guide liquid, heat (regen nozzle)		operating	**high**			
Guide liquid, heat (preburner)		operating	operating		**high**	
Guide liquid, heat (LP turbopump)		operating	operating	operating	operating	
Guide liquid, (Preburner)		operating	operating	operating	operating	
Guide liquid, thrust (Main Combustion Chamber)		operating	operating	operating	operating	
Measure flow		operating	operating	operating	operating	
Measure vibration		operating	operating	operating	operating	

REFERENCES

[1] Roemer M., Byington C., Kacprzynski G. and Vachtsevanos G., "An Overview of Selected Prognostic Technologies with Reference to an Integrated PHM Architecture", *Proc. of the First Intl. Forum on Integrated System Health Engineering and Management in Aerospace*, 2005.

[2] Wu, J., "Liquid-propellant rocket engines health-monitoring—a survey," Acata Astronautica 56(3), 347-356, 2005.

[3] Duncavage, D. Figueroa, F., Holland, R., Schamalzel., *"Integrated System health management (ISHM): Systematic Capability Implementation"* 2006 IEEE Sensors Applications Symposium, Houston, Texas, USA

[4] Orsagh, R.F., Brown, D.W., Kalgren, P.W., Byington, C. S., Hess, A. J., Dabney, T., "Prognostic Health Management for Avionic Systems," IEEEAC paper # 1128

[5] Kurtoglu, T., Tumer, I.Y. *"A Graph-Based Fault Identification and Propogation Framework for Functional Design of Complex Systems,"* ASME Journal of mechanical Design. Vol. 130, May 2008. No 5.

[6] Department of Defense, "Procedures for Performing Failure Mode, Effects, and Criticality Analysis," MIL-STD-1629A.

[7] Vesely, W. E., Goldberg, F. F., Roberts, N.H., and Haasi, D. F., 1981, *The Fault Tree Handbook,* US Nuclear Regulator Comission, NUREG 0492, Washington, DC.

[8] Greenfield, M. A., 2000, "NASA's Use of Quantitative Risk Assessment for Safety Upgrades," *IAAA Symposium*, Rio de Janero, Brazil.

[9] Stamatelatos, M., and Apostolakis, G., 2002, "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners v 1.1," NASA, Safety and Mission Assurance.

[10] Giarratano, J. C., Riley, G. D., 2004, *Expert Systems: Principles and Programming*, 4th ed., PWS, Boston MA, 2004.

[11] Shortliffe, E., 1976, *MYCIN: Computer-Based Medical Consultations*, Elsevier, New York.

[12] Touchton, R. A., 1986, “Emergency Classificaion: A Real Time Expert System Application,” Proceedings of SouthCon.

[13] deKleer, J., and Williams, B. C., 1987, “diagnosing Multiple Faults,” Artif. Intell., 32, pp 97-130.

[14] Chen, J., and Patton, R. J., 1998, *Robust Model-Based Fault diagnosis for Dynamic Systems*, Kluwer Academic, Dordrecht.

[15] Dvorak, D., and Kuipers, B. J., 1989, “Model Based Monitoring of Dynamic Systems,” IJCAI.

[16] Patton, Rr., Frank, P., and Clark, R., 1989, *Fault Diagnosis in Dynamic Systems: Theory and Applications*, Prentice Hall, Hertfordshire, UK.

[17] Kurtoglu, T., Tumer, I. Y., “FFIP: A Framework For Early Assessment of Functional Failures in Complex Systems,” International Conference on Engineering Design, Pairs, France, 2007.

[18] Pahl, G., and Beitx, W., 1984, Engineering Design: A systematic Apprach, Design Council, London.

[19] Kurtoglu, T., Campbell, M. I., Gonzales, J., Bryant, C. R., McAdams, D. A. and Stone, R. B., 2005, “Capturing Empirically Derived Design knowledge for Creating Conceptual Design Configurations,” *Proceedings of DETC2005*, Long Beach, CA, Sep. 24-28.

[20] Wertz, J. R., and Larson, W. J., 1999, *Space Mission Analysis and Design*, 3rd ed., Space Technology Library, Microcosm, Kluwer Academic, Dordrecht.

[21]Sutton, G. P., 1986, *Rocket Propulsion Elements: An Introduction to the Engineering of Rockets*, 5th ed., John Wiley & Sons, New York.

[22] Jensen D., Tumer I.Y., Kurtoglu, T., “Flow State Logic (FSL) for Analysis of Failure Propagation in Early Design” *proc. 21st International Conference on Design Theory and Methodology IDETC/CIE 2009*. San Diego, CA.

[23] Marwell, P., 2003, *Embedded System Design*, Kluwer Academic Publishers, Boston

[24] Kramer, S., Tumer I.Y., “Application of StateCharts and FFIP Methodologies to PHM architecture Co-Design” Submitted to: *Annual Conference of the Prognostics and Health Management Society 2009* San Diego, CA

[25] Tayler, R. N., Hoek, A., “Software Design and Architecture: The Once and Future Focus of Software Engineering” Future of Software Engineering, IEEE 2007

[26] Caporuscio, Georgantas, N., Issarny, V., “A Perspective of the Future of Middleware-based Software Engineering” Future of Software Engineering, IEEE 2007

[27] Lyu, M. R., “Software Reliability Engineering: A Roadmap” Future of Software Engineering, IEEE 2007

[28] Gallardo, M., Martinez, J., Merino, P., Pimentel, E., “On the Evolution of Reliability methods for Critical Software” *Transactions of the Society for Design and Process Science,* Vol. 10, No. 4, December 2006

[29] McKelvin, M. L., Eirea, G., Pinello, C., Kanajan, S., Langiovanni-Vincentelli, A. L., “A Formal Approach to Fault Tree Synthesis for the Analysis of Distributed Fault Tolerant Systems,” *EMSOFT’05*, Sept 19-22, 2005, jersey City, New Jersey, USA

[30] Graham, J. H., “FMECA Control for Software Development,” *Proceedings of the 29th Annual International Computer Software and Applications Conference 2005*

CHAPTER 3

APPLICATION OF STATE CHARTS AND FFIP METHODOLOGIES TO PHM ARCHITECTURE CO-DESIGN

Scott Kramer

Irem Tumer

ABSTRACT

Modern complex systems have evolved into artifacts that rely on both hardware and software to dependably function without human control. Prognostics and Health Management (PHM) systems are developed to manage the risk in these complex systems. As the monetary cost and the potential for loss of life climb as human artifacts strive to do things and go places that have not been done before, these complex systems become reliant on PHM systems to help detect, mitigate, and respond to potential failures. Engineers now incorporate methods to analyze and consider risk and reliability of complex systems from the earliest stages of complex system development. At present, PHM capabilities are only indirectly considered at early stages, if at all. In order to incorporate this vital component of a complex system in its proper place, PHM development should be pushed to the earliest stage of complex system development possible with this change. Engineers can consider PHM capabilities and limitations and make appropriate changes to the overall system earlier in the design stage, where changes are less costly. In previous work, several risk and reliability techniques were discussed, and Function Failure Identification Propagation Framework (FFIP) was selected as best fitting PHM development. In a prior paper, FFIP was augmented to better gather the data needed for PHM implementation. In this paper, this work is extended by taking the data gathered from FFIP and applying a development language often used in the field of embedded systems design. Specifically, the concept of state charts is applied to the FFIP methodology to better and more completely program the Function Failure Logic Reasoner in FFIP by clearly laying out the hierarchical relationships between system health, function health, component status, command signal, and sensor signals. State charts are then applied to the development of a preliminary PHM hardware and software architecture. Additional considerations, such as sensor and software reliability, as well as future considerations are discussed.

INTRODUCTION

PHM systems are responsible for determining the condition of each element in a complex system, detecting anomalies, diagnosing causes, predicting system impact, and initiating appropriate system responses while communicating the appropriate data, information, and knowledge to control architectures and operators in a contextually appropriate manner 12. Since PHM systems permeate throughout the entire complex system and are electro-mechanical in nature, they are essentially embedded hardware-software systems integrated into the larger complex system [3]. The responsibilities of PHM are significant and the PHM architecture is complex. This often results in the PHM development not being addressed much is known about the details of the complex system the PHM architecture will address. As a result, PHM often is retrofitted as loose combination of fault detection and isolation of the various subsystems as opposed to an integrated system-wide application. Historically, PHM has been simple, and limited to sensor technology and minimal computational ability. PHM is often now required to be an integral part of operating complex artifacts that are too complex for human operators to manage without assistance [4]. Many modern complex systems operate without human operators at all, underscoring a greater need for a well integrated PHM system. As such, it is vital to treat PHM as an integral component considered within the design from the earliest possible stages.

As it currently stands, PHM development requires an advanced stage of complex system modeling. Many advanced modeling techniques exist for complex system design and significant and important research continues to produce accurate simulations of complex systems. These simulation processes, however, require advanced stage designs with high levels of detail where many decisions have been made regarding the complex system architecture. Although there is significant work developed for specific PHM applications that can be leveraged at these advanced design stages, any changes that might be made to the complex system as a result of PHM development are costly. Developing a methodology to consider PHM development, PHM capabilities, and PHM limitations earlier in the design stage can highlight elements of the complex system that may require reconsideration or

modification earlier in the design process, where architecture changes have less impact on project cost and schedule. Pushing PHM development to the earliest stages of complex system design also allows the process of testing, validating, and refining PHM architecture to start earlier, allowing a greater opportunity to create a higher quality design.

In previous work, Function Failure Identification Propagation (FFIP) was introduced as a novel way to help with PHM design and was augmented to better capture the data, information, and knowledge needed to begin considering PHM development. This will be reviewed in this work, followed by a methodology to build the PHM architecture through a state charts-based Function Failure Reasoner. Finally, the proposed process will be applied to a Liquid Fueled Rocket Engine (LFRE) model to illustrate the potential benefits.

FFIP for PHM design

The Function Failure Identification Propagation (FFIP) framework, developed by Kurtoglu and Tumer [5,6], models the function, structure, and behavior of systems to simulate failure propagation paths and determine resulting functional failures. Information gleaned from this process can then be used to determine and analyze failure mitigation options in the early design stages of complex systems. Originally, the three major modules used in FFIP analysis are the graphical system modeling, the behavioral simulation, and the function-failure logic (FFL) reasoner. The graphical system modeling includes the functional flow graph (FFG) represents the overall system functions decomposed into smaller fundamental subfunctions. The subfunctional relations are represented through flows of energy, material, and signal (EMS). The structure of the design is represented in the configurational flow graph (CFG). The CFG is linked directly to the FFG and thus strictly adheres to the functional topology of the complex system. Each node of the CFG represents a system component, and the arcs are the same EMS flows as in the FFG. These models are then used to build the Function Failure Logic (FFL) Reasoner, a finite state simulation model founded on basic principles of selected component behavior. The

system is modeled in various operational and failed states in a component oriented approach.

The simulation process is performed to determine system behavior under various failed component conditions. At any given instant, the overall system state is a function of the component modes and system state variables:

$$X(t)=F(c(t),v(t)), \qquad \text{(Equation 1)}$$

where $c(t)=[c1, c2, c3,\ldots,cN]$ is a vector of discrete component modes and $v(t)=[v1,v2,v3,\ldots,vk]$ is a vector of system state variables. The system is modeled as a finite state computational model because quantitative details of the system cannot yet be determined. The quantitative modeling is left for the physics based modeling schemes.

For example, valves can have liquid flow levels of {zero, low, nominal, high} and a sensor may have values of {no signal, high, nominal, low, zero} as the finite state possibilities for these components. These components, and their attached finite state possibilities, are then treated as portable modules that can be fit together into functional models of complex systems. Failed states are entered into the model, and the finite state computations propagate the failures along the functional flows to their ultimate ends, which may or may not result in changes in overall system functionality.

In Kramer and Tumer (2009), FFIP was augmented in three important ways to enable PHM development. The first augmentation, which does not directly concern PHM development in the early stage of design, considers catastrophic failures such as violent explosions. These types of catastrophic failures can propagate across functional flows. A nomenclature was added to the FFG and CFG to identify and track the possibility of these lateral flows across functions. Although there is too much uncertainty to include finite state logic in the FFL reasoner, this provides a means within the visual representations to consider important but not always readily

apparent or computable propagation paths. Eventually, this logic becomes important for the refined PHM architecture in that the PHM system will have to investigate lateral propagation if a catastrophic failure is detected.

The remaining two augmentations are critical for early stage PHM development. The first augmentation is an estimation of time to propagation. Just as components are given finite functioning and failed states in the FFL reasoner, an estimate of how long it takes each failed state to propagate along the functional flow is critical to assessing the ability of PHM to respond to a given failure. Failures that propagate slowly are failures that PHM can be used to help mitigate or correct. Failures that propagate instantly may not give the responses time a PHM system might require to correct or mitigate the problem, and thus engineers should address failures that propagate very quickly with another method. Nomenclatures were added to the FFGs and CFGs capturing these estimations so they can be incorporated in the finite state machine simulation. The second augmentation important for early stage PHM system design addresses explicit PHM response simulation. Although sensors were included in the original FFIP architecture and were part of the initial FFL reasoner, this augmentation addresses the PHM system elements as separate system functions with their own place in the FFL reasoner, instead of being buried in the existing system logic. Addressing the PHM system explicitly in the FFL reasoner allows engineers to gain a better understanding of how PHM is acting on the complex system, and to begin understand the PHM architecture details needed for that particular complex system application.

Contributions

In the previous work, the Function-Failure Logic (FFL) reasoner was coded using finite state logic in a process that ‘hid’ the logic in cell programming in EXCEL. As the FFL reasoner can be extended into a preliminary PHM program specification, this paper focuses on applying embedded system development methodologies to the construction of the FFL reasoner in a manner that graphically exposes the FFL reasoner and PHM response logic.

Also in the previous work, FFIP's CFGs and FFGs only addressed preliminary sensor placement. The FFL reasoner relates sensor signals to PHM response logic through the same 'hidden' finite state machine simulation as above. Application of embedded system design methodologies aims to more clearly and visually demonstrate the hierarchy between sensor signals, component states, failure states, failure propagations, and PHM response.

RELATED RESEARCH

Embedded System Design Specification Development

In this paper, we focus on the software development needs of a PHM architecture. The field of Embedded Systems Design is looked to for guidance. Embedded system design is a sub field of co-design and looks at designing complex hardware and software intensive systems. Co-design is a field that develops compact complex systems such a computers and consumer electronics, the software code is developed in sync with the physical circuitry that runs the code. Often, because of space constraints, the physical size of the circuits, the processing and response speed of the system, and the cost of mass production must be all be optimized at the same time. This can lead to the development of certain application specific circuits in some instances, while off-the-shelf processors may be used elsewhere in the system. The embedded systems sub field addresses the above concerns, but must also address a larger more complex artifact that the embedded system must interact with. This interaction could mean control, response, data collection, or a combination of all three [7, 8,9].

PHM, by definition, is an embedded system. In many modern human artifacts, PHM is a system that controls, responds to, collects data from a larger more complex system. As such, several early stage embedded system design languages are investigated, and the process best fitting the FFIP approach to PHM development is selected and applied to the LFRE.

In embedded system design, as with the design of any complex system, an important starting point is a detailed specification. Although there are a great many languages used in practice, the large majority of them are based on StateCharts, Specification and Description Language (SDL) or Petri nets. StateCharts provide an effective way to demonstrate hierarchy and process oriented computations in a visual manner and will be discussed in further detail. SDL, a finite state machine based process actually designed for distributed system applications, is similar to StateCharts in layout and

logical capabilities, but includes a ‘message passing’ aspect that accounts for many embedded systems not having universal broadcast capabilities and must rely on message queues [10]. Based on the preliminary level data presented in FFIP, SDL may even be too far advanced as system portioning is not yet considered. Also, SDL demonstrates the most usefulness when addressing system efficiency through application specific circuits applications where the embedded system is a large portion of the overall system architecture. For our LFRE application, we are assuming that the finite states are capable of broadcast communications through a central processor for the entire system As PHM for modern complex system applications will be a miniscule fraction of the mass of the system but a significant contribution to system capability, it will probably not fall victim to optimization (at least in terms of central processing capability). If it is deemed necessary at a later stage in the design process, the finite state logic of state charts can be increased in complexity to account for SDL message passing capabilities.

Petri nets provide another specification language developed originally for software but has seen use in embedded systems. Petri nets has the advantage of demonstrating logic unequivocally and completely, but does so at the cost of simplicity[11]. Because Petri nets ‘blow up’ as soon as the system increases in complexity, and because Petri nets do not represent system hierarchy in a clear fashion, they are not considered for use here.

StateCharts

StateCharts is another common early stage language used in embedded systems design. StateCharts is also used as the foundation for other system development methodologies. SpecC and SystemC are both rapidly evolving StateCharts based embedded system and software development processes [12, 13]. Although an industry standard has not emerged from the current fray, most of these methodologies utilize the fundamental strengths of state charts and are running into similar difficulties in their development.

The main strength of StateCharts is the clear hierarchical organization of the computational processes (in the form of finite state machines) required by the larger complex system. Although code cannot be written in a complete form until further along in the design process, an early StateCharts diagram of the program can show developers where the components of the code will fit within the system much in the same way that a configurational diagram shows where the physical components will fit within a larger complex system [14]. Figure 3.1 is a simple visual of the hierarchical arrangement of the functions down to components and the components down to base sensors. Although the finite state logic is not stated here, the relationships between functions, components, and sensor signals are made clear. This logic is extended up through the overall system function capturing the entire operational logic of the complex system. Given enough complexity, this framework can eventually turn directly into code as enough information is added to the architecture.

In a PHM system, the sensors are the ‘tip of the spear’ for the subsystem. As such, the sensors are the lowest level state charts would reach in this architecture. Each component state will be a function of signals gathered in the system. It is not critical that sensors be directly related to this component, but as the sensors are the ‘eyes, ears, nose, and taste buds’ of the PHM system, it is critical that all the information in the PHM system be built on a foundation of sensors and their related signals.

METHODOLOGY

StateCharts applied to the construction of the FFL reasoner.

Demonstrated in figure 3.2 is the modular nature of FFL reasoner logic formation. The top three layers, Functional Model, Configurational Flow Graph, and Function Failure Logic are taken from previous work by Kurtoglu and Tumer (2008). For two types of components, a pipe and a valve, the configuration and function are shown. Below these graphical representations, the FFL logic that corresponds to those components is written in common if-then finite state logic. Each possible finite state combination is listed with a designated output for each combination set. From this, each component in a function stream has a finite impact on the larger function that a particular component serves.

The third layer in the diagram, the StateCharts model, represents the function failure logic in a graphical sense. Each component is given an individual state chart. In that state chart, there is a control signal input, a sensor input, and a finite state output. The single control signal has arrows emanating from it corresponding to the possible signal inputs. The arrows are labeled with 'edge labels' that indicate the finite state corresponding to that logical flow. Each signal input possibility points to either the next piece of information the reasoner needs or to a finite state conclusion. When the arrows lead to the next needed information source, the process repeats until a finite state, on the far right of the state chart, is reached. From that finite state the arrows then lead up the hierarchy to the function above that particular state chart. That function (not shown for space considerations) will have a separate state chart that will have command inputs as well as finite state conditions reported from all subordinate components or functions.

Because the FFIP architecture aims to model the propagation of component failures, the 'input' to the FFL reasoner simulation occurs at the component level. Essentially, all logic arrows will point away from this level of the state charts hierarchy. This results in information being passed up to functional levels resulting in outputs with regard to overall system function, and information being passed down to the sensor

level resulting in outputs that demonstrate the ‘sensor signature’ of that failure. As failures propagate down a functional path, the state of other components are changed through the functional level reasoner, which results in another component change of state input and thus additional expected sensor reading output from the reasoner.

This process differs from the Function Failure Logic previously used in that: 1) the logic is laid out in a visually clear manner, as opposed to hidden in code buried in cells, and 2) these individual components do not determine the state of the overall system functions, but input data to a higher level state chart that determines the state of the higher level functions.

APPLICATION TO PHM DEVELOPMENT

he existing FFIP framework is briefly explained using a moderately complex Liquid Fuel Rocket Engine (LFRE) architecture as a working example. The model presented here is a basic staged combustion model similar to those used in the Space Shuttle operated by NASA. This model is chosen to demonstrate FFIP application to a model that is more complex than what has been used in the existing literature but still very much a conceptual design. In this model, liquid hydrogen is used as both the fuel as well as the coolant. The first pressure boost the hydrogen flow receives comes in the low pressure fuel booster, which is driven by hot gasified hydrogen carrying heat energy from the regeneratively cooled tubular nozzle. A coolant control valve regulates the amount of hydrogen that flows through the regeneratively cooled nozzle sending the hydrogen that is not used for cooling directly to the preburners. The hydrogen is combusted in preburners with some of the oxygen that drive turbopumps on both the fuel and oxidizer flows. The oxygen low pressure turbopump is driven by liquid oxygen pressurized by the high pressure oxygen turbopump that is connected to the oxygen side preburner. Some of the oxygen is burned in the preburners and the remainder is injected into the main combustion chamber where it burns with the hot gasses generated in the preburners [15].

The configurational flow graph (Fig. 3) lists the components and the flows between them. The configurational flow graph maps directly to the functional flow graph (Fig. 4) where the functions of the system are clearly laid out. The flows delineate the transfer of signal, energy, and material between the different functions and components. In prior FFIP applications in the literature, the flows of signal, energy, and material are considered separately, but in this example, some of the hydrogen flow and some of the oxygen flow is represented as both energy and material in the same flow as energy is carried back to the low pressure turbopumps by these material flows.

Using the finite state logic developed earlier (Fig. 1), a Function Failure Logic (FFL) reasoner is developed that reflects this LFRE system. Although the finite details about the operating parameters of this system have not been developed, a simplified early stage simulation tool is developed that will show how the failures of components within the system will propagate through the system and potentially affect a change in the overall system state.

Figure 3.5 demonstrates that the Guide Liquid function for the fuel flow is a function of 8 different components. One of those components is the inlet valve, which is a function of command signals, the fuel flow meter, and the heuristics of the remaining fuel flow sensors. The solid black input ports indicate a base signal, and the hollow circles indicate an output from a sub function below the Inlet Valve component state. The coolant control valve is another component inputting into the Guide Fuel function with associated sensor signals. The remaining components listed have similar state charts developed. The Coolant Control Valve state chart lists the finite states that are input to the state chart and the finite state outputs, sent up the state charts hierarchy as well as the expected control signal and sensor states that are down a level in the hierarchy.

A key set of information drawn from this state charts look is a early assessment of sensor placement effectiveness. As each function of the system has a specific state charts hierarchy, we can see that determining the health of each function that is ultimately reliant on the system sensors and command inputs. Although not explicitly listed here for space concerns, one can readily see that the ‘Guide Liquid’ function can be followed down the state charts hierarchy to a sensor ‘fingerprint’ for the various finite states of that particular function. Already, PHM designers can see if each function state has a unique fingerprint for the PHM system to read and assess. This may require more sensors in specific locations while other instances may reveal excess information and allow for the reduction of sensors.

In the edge labels of the Coolant Control Valve state chart, one can see that the failed states also have numbers next to them. These demonstrate the Time to Failure Propagation discussed in earlier work. Again, these are 'order of magnitude' estimates as to how quickly a failure propagates down stream. A '1' indicates slow propagation, '2' is moderate, and '3' is rapid. These estimates are important for two reasons. First, they help to differentiate between failures that PHM can be expected to respond to, and second, they add a small amount of additional detail about system behavior. When StateCharts architectures are applied to embedded system designs, there are functions that imbed a delay into the process, such as the delay between receipt of phone call and when an answering machine picks up. For FFL simulation, a slow propagating failure can have the propagation delayed by a few time steps giving designers a more accurate look at system behavior before delving into high detail physics based models.

As this is a preliminary look at the PHM architecture from an early stage, the details such as timing, message passing, clock usage, and other more detailed tools of state charts applications are not yet applied in a final state. Time is addressed, but only in an order of magnitude fashion as a tool for estimating PHM response abilities.

DESIGNING PHM SYSTEM RESPONSE

With the system modeled as a finite state machine that builds the FFL, engineers than can address PHM response through building further on the StateCharts model. So far, we have demonstrated that StateCharts can be used to translate the hierarchy of the complex system into an embedded systems specification language. This representation clarifies the finite state cause and effect relationships between the system, the functions, sub functions, individual components, and the sensors with which the PHM 'sees' the system. This is important to building the FFL reasoner that is an early state system simulation used to study failure propagation and impact on larger functional structures.

PHM response, however, cannot fit within the direct hierarchy of the system simulation as it acts throughout the entire complex system through its embedded nature. The responses initiated by PHM will come down the hierarchy in the form of command signals, and the input to the PHM response will come up through the hierarchy in the form of sensor signals. As such, the PHM sub system state charts will exist 'next to' the system simulation hierarchy (Fig. 6).

When a failure is noted in the existing state charts architecture and a PHM response is identified, the state chart for that component will also send a message over to the PHM response hierarchy (Fig 7). These StateCharts, based on responses determined through the FFIP analysis process as well as other failure analysis techniques, will represent the automatic responses the PHM system will take within the overall system [16]. This logic should be separate from the finite state logic already built into the system so as to continue to analyze the system logic with and without PHM response.

Before the FFIP process can be evolved into a StateCharts based preliminary embedded system design specification from the FFL reasoner, it is worth noting that

the relationship between expected state values and expected sensor outputs is made clear once again. Although the engineers designing the system have intuition, experience, and imagination to rely on for identifying failures, the PHM system has only sensor signals. This once again underscores the idea of a failure having a 'sensor fingerprint' that is different from other failures, thus enabling a PHM program to identify the specific failure and enact the correct response.

This is also a stage that can make extensive use of other established early design risk and reliability techniques. Specifically, this is a stage where engineers must brain storm about potential failures and then ask 'what do we want to do to respond to, mitigate, or even ignore this failure'. Mitigation may require partial or complete system redesign, but if engineers determine that system response is the correct action, either the embedded PHM system or the operators will be responsible for initiating that response. Currently, in industry, FMECA is used to build PHM logic through defining needs and coding to specific scenarios. As PHM grows, it is likely that PHM will be responsible for detecting the failure and initiating response without operator command. These responses have their programming start here, at the conceptual design stage in the PHM response state charts.

In previous work to augment FFIP for implementation to LFRE design, PHM response was discussed for handling a slow propagating valve failure. In the LFRE system described above, the coolant control valve was simulated 'failed closed' forcing all the fuel to flow through the regeneratively cooled tubular nozzle. As the fuel then acts as a material and energy transport carrying thermal energy from the nozzle back up to the low pressure fuel booster turbopump, the more fluid going through the cooling apparatus on the nozzle carries more thermal energy to the fuel booster turbopump. The additional energy will drive the pump faster thus pushing more fuel into the system. With more fuel pushed in, the fuel-oxygen ratios will be different than design parameters. At this stage it is impossible to determine exactly how the system would

respond to the coolant control valve failure, but we can confidently predict that the change in fuel flow will cause a change in vibrations within the system, and that system failures due to vibration can be catastrophic, but usually require time to propagate. As the failure has potentially significant consequences, and there is time enough for PHM to respond, engineers can employ mitigation expertise to program a PHM response (Fig 8). In this case, we chose to regulate the fuel inlet valve. Although more energy is being carried to the fuel booster tubropump, the fuel flow is regulated to nominal levels with another valve. This response is fed into the upper levels of the state charts hierarchy in the form of a command signal changing the state of a component. If the state charts are programmed correctly, this change of state propagates through the state charts hierarchy resulting in nominal fuel mixtures and thus nominal vibrations.

RELIABILITY CONSIDERATIONS

State charts applied to sensor reliability and effectiveness

Knowing the fingerprint of each state chart can allow for assessment of components that may not have direct sensor data. By using data from elsewhere in the system, understanding the hierarchal relationships between components and system functions, and understanding how failures will propagate through a system, an intelligent PHM can determine the state of a given component without a direct reading from that component through virtual sensors.

As each failure mode will have a 'fingerprint' composed of sensor readings informing the PHM system as to the health of the components involved in each function, engineers can begin to address fingerprints that are over informed, where there may be excess sensor data, and to address fingerprints that are under informed, where there is not enough sensor data. From an early stage, PHM hardware efficiency becomes a consideration at an early stage and can inform the efficiency of the overall system design.

State Charts modular programming construct can address the development of such sub routines much earlier in the design stage. As soon as a sensor type is selected for a specific application, software engineers can begin to consider the computational needs of that set of sensor arrangements. Practically, there is extensive research done in the field of sensor signal interpretation for many complex systems, and the search for matching existing work can begin during the FFIP analysis stage if the software development of PHM development is also developed at this point. Given that many systems will have specific complex heuristic needs that are not yet developed, applying a state charts paradigm to the FFIP process addresses these needs earlier in the design process.

Here, sensor reliability can also begin to be addressed. Although this process will not inherently improve sensor reliability, it can be used to account for reliability issues early on. Within the heuristics of each state chart, allowances can be made for the known reliability levels of sensors. As sensors rely on a huge range of technologies, and there is data pertaining to sensor types and their associated reliability levels, engineers can begin to address the risk and reliability analysis of the PHM sub system hardware. This knowledge can greatly reduce the amount of reactions based on faulty sensor readings, and increase the odds of correctly recognizing and diagnosing a failure that may not fit the entire state charts 'fingerprint' for that functional failure. In essence, as FFIP is used to address risk and reliability of a complex system, and FFIP is demonstrated as a tool to build a PHM system that addresses system reliability and response to failures, this process applies the same principles to the PHM subsystem itself.

As stated above, sensors form the foundational level of the state charts hierarchy. As it stands, FFIP considers the physical components and some of the signals required to control those physical components. A PHM system requires more information, even at an early design stage. As such, it is not difficult for experienced engineers to guess where sensors might be placed in a complex system, even at the earliest stages of design. An experience based starting point is sufficient to begin the FFIP based PHM software design and analysis through a state charts based architecture.

Addressing software reliability through State Charts methods.

The process discussed thus far is based on FFIP, which addresses the potential hardware component failures early in a complex system design. The rapidly evolving field of robust software design also employs methods that address potential software failures and the subsequent results of those failures while early in the design process. Methodologies are employed that address software failure from early specification development through testing with simulation platforms all the way through to final

product validation [17,18,19,20,21,22]. Some of these processes are modeled on and in many ways emulate processes applied to mechanical systems. Regardless of which process is used, this state charts based PHM language development provides a starting point for addressing the reliability of the software side of any given PHM design. In fact, many embedded systems development processes employ running the programs through specifically designed testing platforms. As software reliability research has only found methods to find failures as opposed to confirm the absence of potential failures in software code, the earlier the validation process can begin, the better.

CONCLUSIONS AND FUTURE WORK

In this work, Function Failure Identification Propagation has been taken a step beyond a failure propagation analysis tool and been transformed into a PHM design template. Through the addition of embedded systems design fundamentals, a process developed for complex system mechanical interactions can now consider directly the complex computational architectures required to monitor and respond to failures in complex systems. The evolution of complex systems is quickly shaping most modern complex systems into electromechanical systems highly dependent on computerized operations. In this work, FFIP is transformed into a risk and reliability analysis tool capable of modeling and capturing the fundamental consequences of failures in these systems from an early design stage, where design corrections are cheapies and easiest.

Although this research has focused on PHM design based upon failure analysis methodology, the rapidly evolving field of unified modeling languages, such as SYSML and SystemC and SpecC previously mentioned, should be the future tool used to design systems complex enough to require PHM architectures. As complex mechanical systems become more reliant on programming for control and response, consideration of the programming architecture at a very early stage becomes inescapable. This work, in many ways, is a step in that trend and it has transitioned from considering the physical architecture of a complex mechanical system to also considering the computational architecture in monitoring the health of that complex system. Extrapolation to incorporating the entire command and control architecture into this process is well within the realm of possibility. The process with which to do that resides in unified modeling methodologies. As is the case with much in embedded system language development, the field is moving quickly and there are several processes that may eventually establish as industry or universal standards. The work herein is generalized enough to fit into many of those processes as is. Challenges come as the design becomes more refined and application specific needs become clear, such as mixed signal allocations, message passing for distributed systems, application specific circuit needs.

TABLES & FIGURES

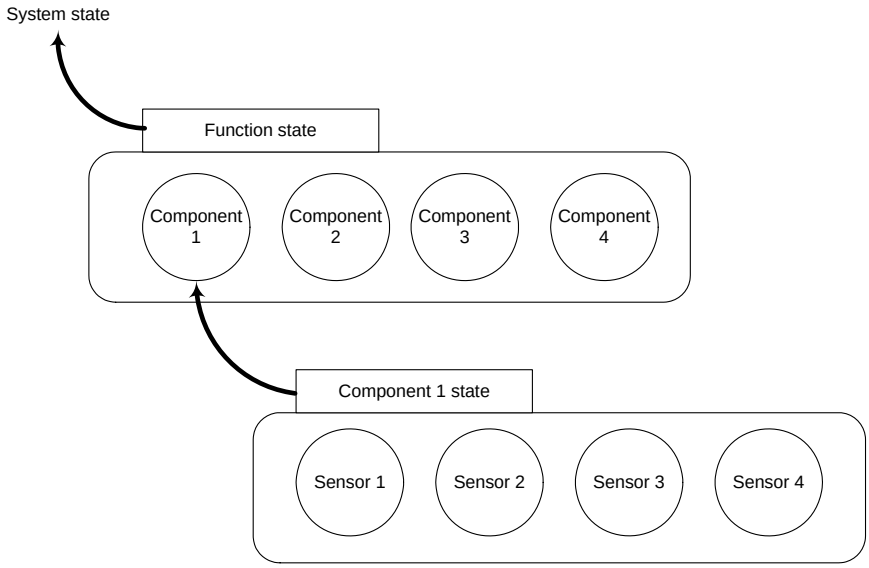

Figure 3.1: StateCharts heirarchy.

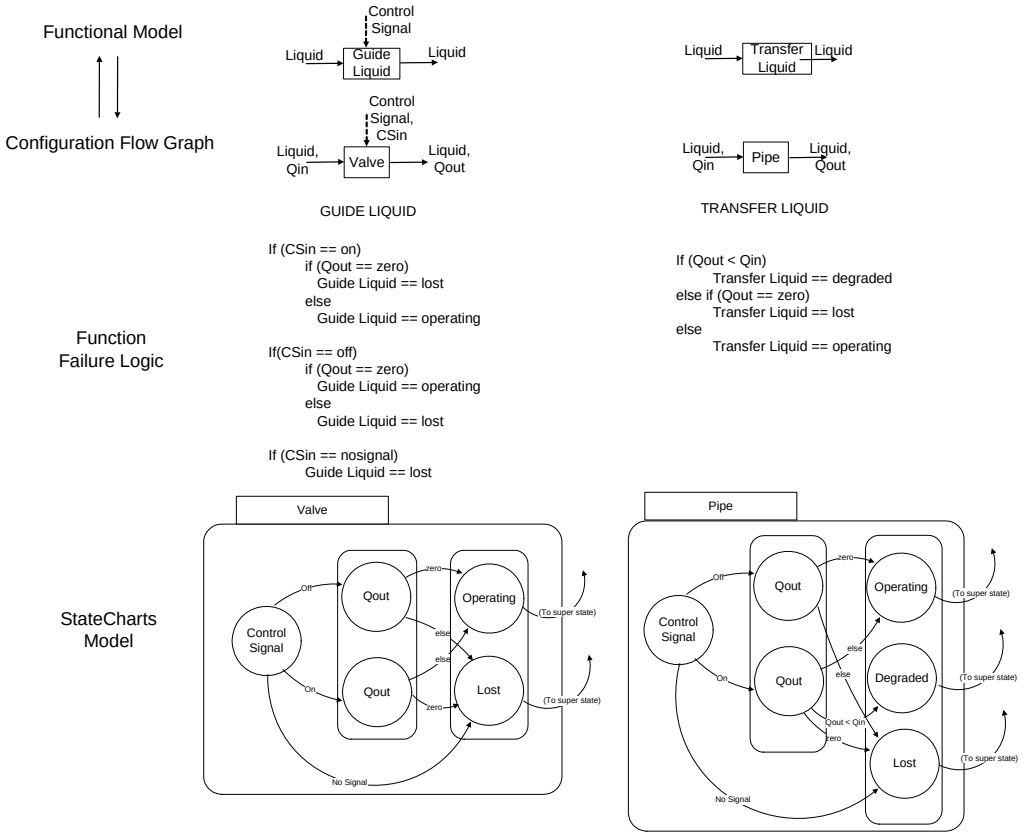

Figure 3.2: StateCharts applied to component logic

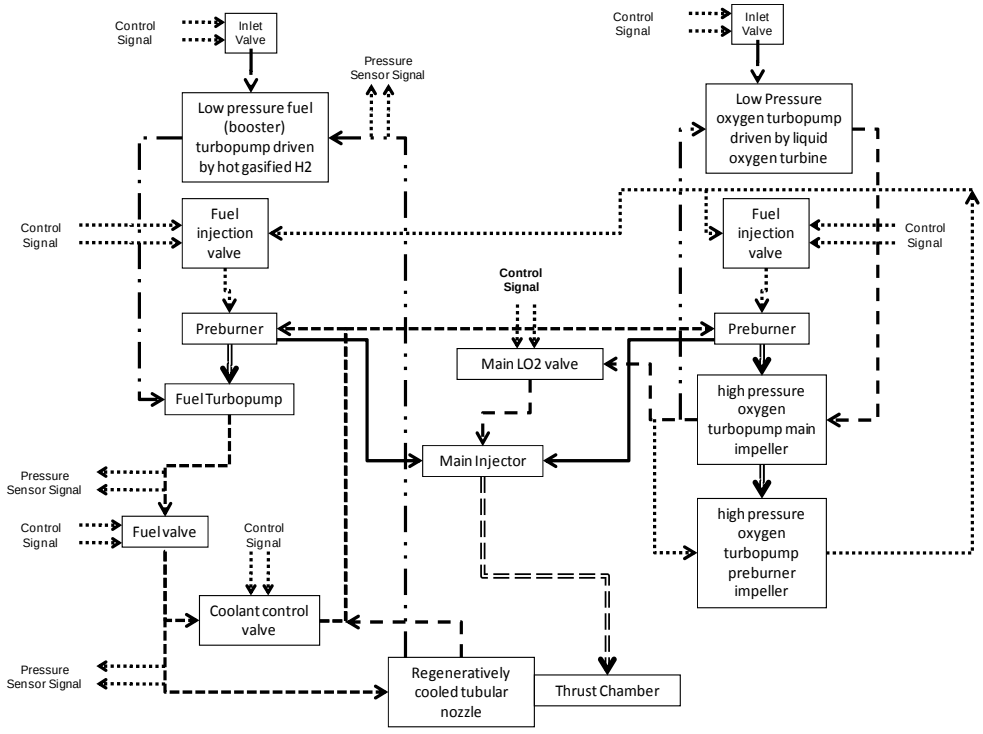

Figure 3.3: Configurational Flow Graph for the Liquid Fuel Rocket Engine

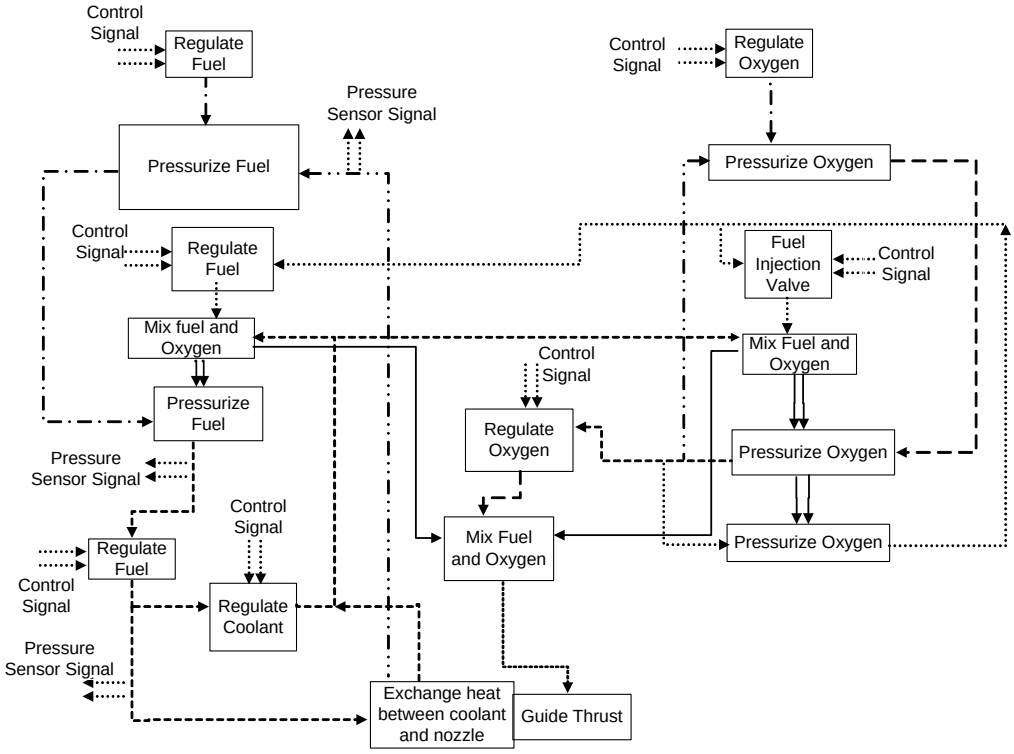

Figure 3.4: Functional Flow Graph for the liquid Fuel Rocket Engine.

Table 3.1: FFL logic applied to the Liquid Fuel Rocket Engine coolant valve failure.

	at t=0	at t=1	at t=3	at t=4	at t=5
Component Modes					
Inlet Valve	nominal	nominal	nominal	nominal	**high**
Low Pressure Fuel Booster	nominal on	nominal on	nominal on	**high**	
Fuel turbopump	nominal on	nominal on	nominal on	nominal on	**high**
Fuel valve	nominal on	nominal on	nominal on	nominal on	**nominal on**
Fuel valve controller	nominal on	nominal on	nominal on	nominal on	nominal on
Coolant control valve	nominal	**failed off**			
Coolant valve controller	nominal on	nominal on	nominal on	nominal on	nominal on
Regeneratively cooled tubular nozzle	nominal	nominal	**high**		
Preburner	nominal on	nominal on	nominal on	nominal on	nominal on
Low pressure turbopump	nominal on	nominal on	nominal on	nominal on	**high**
Preburner	nominal on	nominal on	nominal on	nominal on	**degraded**
Main combustion Chamber	nominal	nominal	nominal	nominal	**degraded**
sensor	nominal on	nominal on	nominal on	nominal on	nominal on
State Variables					
Liquid Flow	nominal	nominal	nominal	nominal	**high**
Pressure increase	nominal	nominal	nominal	**high**	
Liquid Flow	nominal	nominal	nominal	**high**	
Pressure increase	nominal	nominal	nominal	nominal	nominal
Liquid Flow	nominal	nominal	**zero**		
Pressure control	nominal	**zero**			
Liquid Flow	nominal	nominal	**high**		
Combustion	nominal	nominal	nominal	nominal	**degraded**
Liquid Flow (Pipe 6)	nominal	nominal	nominal	**high**	
Liquid Flow (Pipe 7)	nominal	nominal	nominal	**high**	
Control Signal Flow	nominal	nominal	nominal	nominal	nominal
Status Signal Flow	nominal	nominal	nominal	nominal	nominal
System Functions					
Import Liquid (inlet pipe)		operating	operating	operating	operating
Guide Liquid (LP)		operating	operating	operating	operating
Guide Liquid (Turbopump)		operating	operating	operating	operating
Guide Liquid (Fuel Valve)		operating	operating	operating	operating
Guide Liquid (Coolant control valve)		**lost**			
Process signal (Coolant valve controller)		operating	operating	operating	operating
Guide liquid, heat (regen nozzle)		operating	**high**		
Guide liquid, heat (preburner)		operating	operating		**high**
Guide liquid, heat (LP turbopump)		operating	operating	operating	operating
Guide liquid, (Preburner)		operating	operating	operating	operating
Guide liquid, thrust (Main Combustion Chamber)		operating	operating	operating	operating
Measure flow		operating	operating	operating	operating
Measure vibration		operating	operating	operating	operating

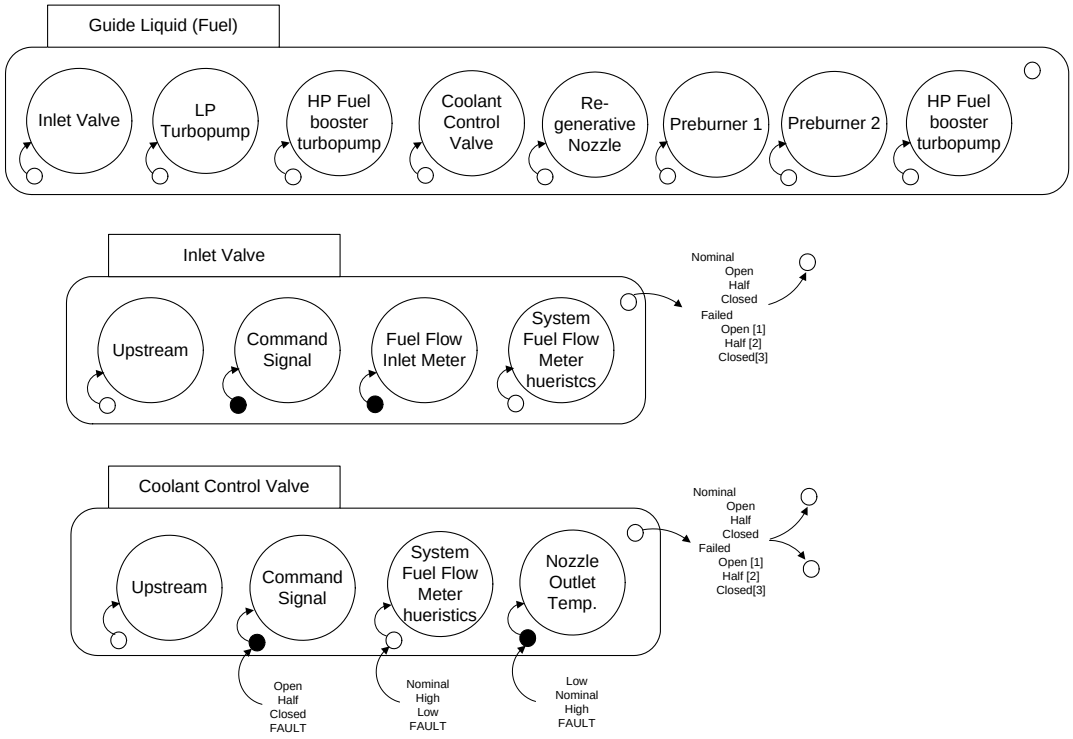

Figure 3.5: Partial StateCharts representation of Guide Liquid functional hierarchy.

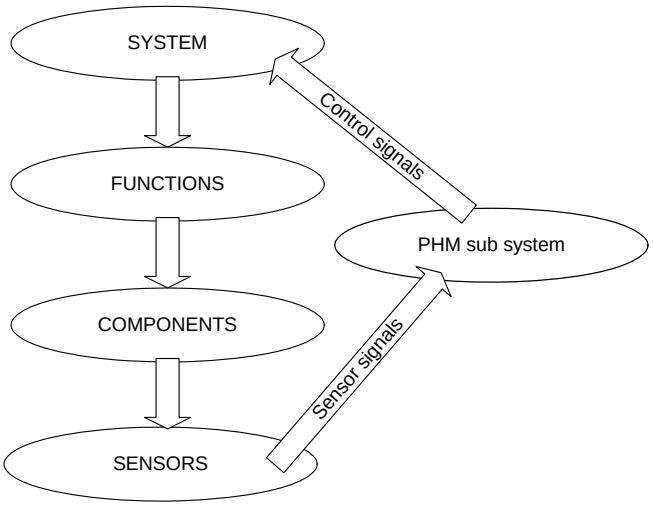

Figure 3.6: A PHM system will interact with the overall complex system.

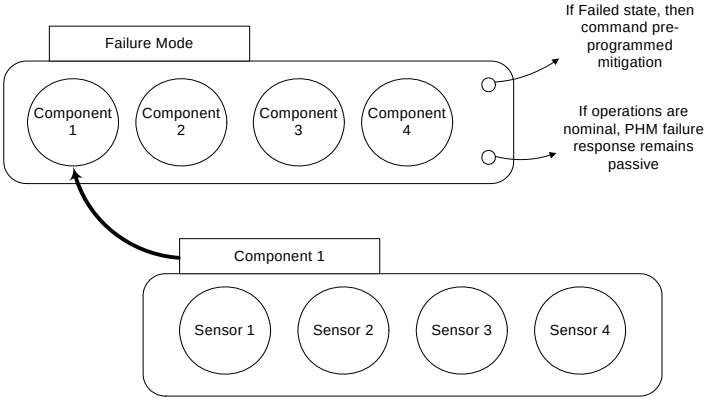

Figure 3.7: StateCharts representation of PHM system computational architecture.

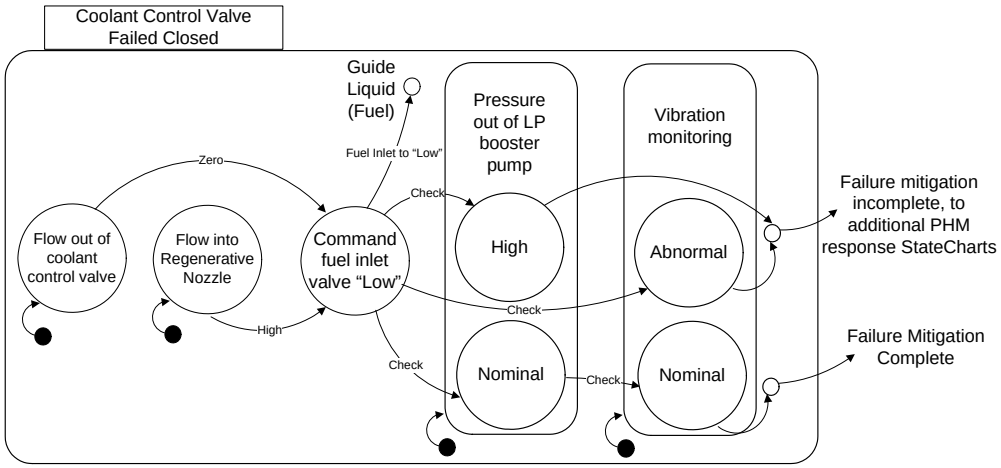

Figure 3.8: StateCharts applied to PHM response logic of coolant control valve failure example.

REFERENCES

[1] Wu, J., “Liquid-propellant rocket engines health-monitoring—a survey,” Acata Astronautica 56(3), 347-356, 2005.

[2] Duncavage, D. Figueroa, F., Holland, R., Schamalzel., “Integrated System health management (ISHM): Systematic Capability Implementation” 2006 IEEE Sensors Applications Symposium, Houston, Texas, USA

[3] Edwards, S., Lavango, L., Lee, E., Sangiobanni-Vincentelli. “Design of Embedded Systems: Formal Models, Validation, and Synthesis” *Proceedings of the IEEE, Vo.l 85, NO 3, 1997.*

[4] Orsagh, R.F., Brown, D.W., Kalgren, P.W., Byington, C. S., Hess, A. J., Dabney, T., “Prognostic Health Management for Avionic Systems,” IEEEAC paper # 1128

[5] Kurtoglu, T., Tumer, I.Y. “A Graph-Based Fault Identification and Propogation Framework for Functional Design of Complex Systems,” *ASME Journal of mechanical Design*. Vol. 130, May 2008. No 5.

[6] Kurtoglu, T., Tumer, I. Y., “FFIP: A Framework For Early Assessment of Functional Failures in Complex Systems,” *International Conference on Engineering Design*, Pairs, France, 2007.

[7] Zave, P. “An Operational Approach to Requirements Specification for Embedded Systems”. *IEEE Transactions on Software Engineering*, May 1982.

[8] Ernst, R. “Codesign of Embedded Systems: Status and Trends” *IEEE Design & Test of Computers*, 1998

[9] Gajski, D., et al *Specification and Design of Embedded Systems* Kluwer Academic, 1994

[10] Marwell, P., 2003, *Embedded System Design*, Kluwer Academic Publishers, Boston

[11] Peterson, J. *Petri Net Theory and the Modeling of Systems*, Prentice Hall 1981

[12] Fujita, M. Hakamura, H. “The Standard SpecC Language” *Proceedings of the ISSS, 2001*

[13] IEEE Standard SystemC, 2006

[14] D. Harel et al. “On the Formal Semantics of Statecharts.” *Proceedings of the 2nd IEEE Symposium of Logic in Computer Science*, 1987

[15] Sutton, G. P., 1986, *Rocket Propulsion Elements: An Introduction to the Engineering of Rockets*, 5th ed., John Wiley & Sons, New York.

[16] Harel, D. et al. “STATEMATE: A Working Environment for the Development of Complex Reactive Systems” *IEEE Transactions on Software Engineering, Vol. 16, N0. 4, 1990*.

[17] Tayler, R. N., Hoek, A., “Software Design and Architecture: The Once and Future Focus of Software Engineering” *Future of Software Engineering*, IEEE 2007

[18] Caporuscio, Georgantas, N., Issarny, V., “A Perspective of the Future of Middleware-based Software Engineering” *Future of Software Engineering*, IEEE 2007

[19] Lyu, M. R., “Software Reliability Engineering: A Roadmap” *Future of Software Engineering*, IEEE 2007

[20] Gallardo, M., Martinez, J., Merino, P., Pimentel, E., “On the Evolution of Reliability methods for Critical Software” *Transactions of the Society for Design and Process Science*, Vol. 10, No. 4, December 2006

[21] McKelvin, M. L., Eirea, G., Pinello, C., Kanajan, S., Langiovanni-Vincentelli, A. L., “A Formal Approach to Fault Tree Synthesis for the Analysis of Distributed Fault Tolerant Systems,” *EMSOFT’05*, Sept 19-22, 2005, jersey City, New Jersey, USA

[22] Graham, J. H., “FMECA Control for Software Development,” *Proceedings of the 29th Annual International Computer Software and Applications Conference 2005*

CHAPTER 4

GENERAL CONCLUSION

In the work discussed here, the objectives of AFSOR grant are addressed through the use of existing tools that have been modified and for new applications. The processes take into account early stage complex system modeling, and emphasize analysis of failure identification and propagation. Based on the functional and structural topology of the early system architecture, these models map between the system's functional structure, the system's physical component layout, the system's behavioral reasoning, and the software-hardware architecture that monitors and responds to failures in the complex system.

By applying a more nuanced and complex failure mode to the concepts developed here, a specification methodology for intelligent PHM system development during early stage complex system design is developed.

Although all the requirements of the AFSOR grant are addressed, they are not yet completely satisfied. The grant, as written, is to be addressed through two fields of engineering research. One, mechanical engineering, is addressed well in these works. The failure analysis methods in common practice are discussed and Function Failure Identification Propagation is selected as best fitting the grant objectives, and then expanded to better fit the grant requirements.

The second field of engineering research is computer science. In this work, the extension of FFIP to consider StateCharts methodologies to address software functionality and requirements. StateCharts representations clarify functional hierarchies, address the importance of sensor placement, and bring into view critical considerations regarding Prognostics Health Management system capability early in complex system design. Software failure analysis, and complete PHM system simulation, however, has yet to be accomplished.

As per the grant's estimated timeline, the research conducted here fits well with what was expected during the first two years of active research, and was conducted from the mechanical engineering field of research. Many of the concepts were implemented in the design modeling architectures of visual natures, and most of them were

implemented in computer programming architectures. Continued work, either into the field of computer science, or with the help of a computer science expert, is needed to fully vet and implement these methodologies into a robust programming architecture. Although the concepts are here, the expertise level in computer science to take on these tasks has not yet been reached in the first two years of grant sponsored research.

Continued work is recommended to carry these methodologies to a full implementation and robust simulation within the field of computer science. Doing so will also elevate the work within from conference worthy research to full journal publication research.

BIBLOGRAPHY

Roemer M., Byington C., Kacprzynski G. and Vachtsevanos G., “An Overview of Selected Prognostic Technologies with Reference to an Integrated PHM Architecture”, *Proc. of the First Intl. Forum on Integrated System Health Engineering and Management in Aerospace*, 2005.

Wu, J., “Liquid-propellant rocket engines health-monitoring—a survey,” Acata Astronautica 56(3), 347-356, 2005.

Duncavage, D. Figueroa, F., Holland, R., Schamalzel., *“Integrated System health management (ISHM): Systematic Capability Implementation”* 2006 IEEE Sensors Applications Symposium, Houston, Texas, USA

Orsagh, R.F., Brown, D.W., Kalgren, P.W., Byington, C. S., Hess, A. J., Dabney, T., “Prognostic Health Management for Avionic Systems,” IEEEAC paper # 1128

Kurtoglu, T., Tumer, I.Y. *“A Graph-Based Fault Identification and Propogation Framework for Functional Design of Complex Systems,*” ASME Journal of mechanical Design. Vol. 130, May 2008. No 5.

Department of Defense, “Procedures for Performing Failure Mode, Effects, and Criticality Analysis,” MIL-STD-1629A.

Vesely, W. E., Goldberg, F. F., Roberts, N.H., and Haasi, D. F., 1981, *The Fault Tree Handbook*, US Nuclear Regulator Comission, NUREG 0492, Washington, DC.

Greenfield, M. A., 2000, “NASA’s Use of Quantitative Risk Assessment for Safety Upgrades,” *IAAA Symposium*, Rio de Janero, Brazil.

Stamatelatos, M., and Apostolakis, G., 2002, “Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners v 1.1,” NASA, Safety and Mission Assurance.

Giarratano, J. C., Riley, G. D., 2004, *Expert Systems: Principles and Programming*, 4th ed., PWS, Boston MA, 2004.

Shortliffe, E., 1976, *MYCIN: Computer-Based Medical Consultations*, Elsevier, New York.

Touchton, R. A., 1986, “Emergency Classificaion: A Real Time Expert System Application,” Proceedings of SouthCon.

deKleer, J., and Williams, B. C., 1987, “diagnosing Multiple Faults,” Artif. Intell., 32, pp 97-130.

Chen, J., and Patton, R. J., 1998, *Robust Model-Based Fault diagnosis for Dynamic Systems*, Kluwer Academic, Dordrecht.

Dvorak, D., and Kuipers, B. J., 1989, “Model Based Monitoring of Dynamic Systems,” IJCAI.

Patton, Rr., Frank, P., and Clark, R., 1989, *Fault Diagnosis in Dynamic Systems: Theory and Applications*, Prentice Hall, Hertfordshire, UK.

Kurtoglu, T., Tumer, I. Y., “FFIP: A Framework For Early Assessment of Functional Failures in Complex Systems,” International Conference on Engineering Design, Pairs, France, 2007.

Pahl, G., and Beitx, W., 1984, Engineering Design: A systematic Apprach, Design Council, London.

Kurtoglu, T., Campbell, M. I., Gonzales, J., Bryant, C. R., McAdams, D. A. and Stone, R. B., 2005, “Capturing Empirically Derived Design knowledge for Creating Conceptual Design Configurations,” *Proceedings of DETC2005*, Long Beach, CA, Sep. 24-28.

Wertz, J. R., and Larson, W. J., 1999, *Space Mission Analysis and Design,* 3rd ed., Space Technology Library, Microcosm, Kluwer Academic, Dordrecht.

Sutton, G. P., 1986, *Rocket Propulsion Elements: An Introduction to the Engineering of Rockets*, 5th ed., John Wiley & Sons, New York.

Jensen D., Tumer I.Y., Kurtoglu, T., “Flow State Logic (FSL) for Analysis of Failure Propagation in Early Design” *proc. 21st International Conference on Design Theory and Methodology IDETC/CIE 2009*. San Diego, CA.

Marwell, P., 2003, *Embedded System Design,* Kluwer Academic Publishers, Boston

Kramer, S., Tumer I.Y., “Application of StateCharts and FFIP Methodologies to PHM architecture Co-Design” Submitted to: *Annual Conference of the Prognostics and Health Management Society 2009* San Diego, CA

Tayler, R. N., Hoek, A., “Software Design and Architecture: The Once and Future Focus of Software Engineering” Future of Software Engineering, IEEE 2007

Caporuscio, Georgantas, N., Issarny, V., “A Perspective of the Future of Middleware-based Software Engineering” Future of Software Engineering, IEEE 2007

Lyu, M. R., “Software Reliability Engineering: A Roadmap” Future of Software Engineering, IEEE 2007

Gallardo, M., Martinez, J., Merino, P., Pimentel, E., “On the Evolution of Reliability methods for Critical Software” *Transactions of the Society for Design and Process Science,* Vol. 10, No. 4, December 2006

McKelvin, M. L., Eirea, G., Pinello, C., Kanajan, S., Langiovanni-Vincentelli, A. L., “A Formal Approach to Fault Tree Synthesis for the Analysis of Distributed Fault Tolerant Systems,” *EMSOFT’05*, Sept 19-22, 2005, jersey City, New Jersey, USA

Graham, J. H., “FMECA Control for Software Development,” *Proceedings of the 29th Annual International Computer Software and Applications Conference 2005*

Edwards, S., Lavango, L., Lee, E., Sangiobanni-Vincentelli. “Design of Embedded Systems: Formal Models, Validation, and Synthesis” *Proceedings of the IEEE, Vo.l 85, NO 3, 1997.*

Zave, P. “An Operational Approach to Requirements Specification for Embedded Systems”. *IEEE Transactions on Software Engineering*, May 1982.

Ernst, R. “Codesign of Embedded Systems: Status and Trends” *IEEE Design & Test of Computers*, 1998

Gajski, D., et al *Specification and Design of Embedded Systems* Kluwer Academic, 1994

Peterson, J. *Petri Net Theory and the Modeling of Systems*, Prentice Hall 1981

Fujita, M. Hakamura, H. “The Standard SpecC Language” *Proceedings of the ISSS, 2001*

IEEE Standard SystemC, 2006

D. Harel et al. “On the Formal Semantics of Statecharts.” *Proceedings of the 2nd IEEE Symposium of Logic in Computer Science,* 1987

Harel, D. et al. “STATEMATE: A Working Environment for the Development of Complex Reactive Systems” *IEEE Transactions on Software Engineering, Vol. 16, N0. 4, 1990*.