Towards StateCharts Based Failure Propagation Analysis for Designing Embedded PHM Systems

Scott Kramer [1], **Irem Y. Tumer** [2]

[1] *Graduate Research Assistant, Oregon State University, Corvallis, OR, 97331, USA*
kramer@engr.orst.edu
[2] *Associate Professor, Oregon State University, Corvallis, OR, 97331, USA*
irem.tumer@oregonstate.edu

ABSTRACT

Modern complex systems have evolved into artifacts that rely on both hardware and software to dependably function without human control. Health management software control systems have been developed to manage failures in such complex systems. The Prognostics and Health Management (PHM) systems have also developed to detect and identify failures and support operation by guiding either operator or automated software response. Of growing of importance in the PHM community is the need to develop formal methodologies to help integrate PHM into the system architecture during the early design stages. Early integration provides designers with the potential to consider PHM capabilities and limitations and make appropriate changes to the overall system earlier in the design stage, where changes are less costly and more effective. In previous work, the Function Failure Identification Propagation Framework (FFIP) was introduced as a novel methodology to help with early design of PHM systems, followed by several required augmentations to make FFIP more effective for PHM design specifically. In this paper, this research 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 StateCharts from the embedded systems design field is used to further augment the FFIP methodology to more completely program the Function Failure Logic (FFL) reasoner module within FFIP. StateCharts are shown to augment the FFIP framework by clearly laying out the hierarchical relationships between system health, function health, component status, command signal, and sensor signals. StateCharts are then applied to the development of a preliminary PHM hardware and software architecture using a liquid fuel rocket engine as a working example. Additional considerations, such as sensor and software reliability, as well as future considerations are discussed.

1 INTRODUCTION

PHM systems are responsible for determining the condition of critical elements 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 (Edwards *et al.*, 1997; Duncavage *et al.*, 2006; Hoyle *et al.*, 2009; Hutcheson and Tumer, 2005b; 2005a; Tumer, 2005; M. *et al.*, 2005). PHM systems permeate throughout the entire complex system and are essentially embedded hardware-software systems integrated into the larger complex system (Edwards *et al.*, 1997). The responsibilities of PHM are significant and the PHM architecture can be very complex. This often results in the PHM development not being addressed until much is known about the details of the complex system into which the PHM system will be integrated. As a result, PHM is often retrofitted as a loose combination of fault detection and isolation of the various subsystems as opposed to an integrated system-wide application. 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. 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 to be considered within the system design from the earliest possible stages.

This paper takes the position that PHM is an integral par of the software control subsystem. As with the software control, PHM development currently requires precise system specification. Many advanced modeling techniques exist for complex system design and significant and important research continues to produce accurate simulations of such systems and their components. These simulation processes, however, require high levels of detail where many decisions have been made regarding the system architecture, components, etc. Furthermore, because many of the deci-

sions have been made at this later design stage, 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.

Efforts to formalize this process have been introduced in the PHM community (Hoyle *et al.*, 2009; Hutcheson and Tumer, 2005b; 2005a; Tumer, 2005; Kurtoglu *et al.*, 2008; Leao *et al.*, 2008). However, these methods stop short of providing a true integration capability, which requires the consideration of multiple faults and the cumulative effect of fault propagation. The Function Failure Identification Propagation (FFIP) framework was introduced in prior work 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 (Kurtoglu and Tumer, 2008a; 2008b; Jensen *et al.*, 2008; 2009; Kramer and Tumer, 2009). In this paper, this research 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 StateCharts is used to further augment the FFIP methodology to more completely program the Function Failure Logic (FFL) reasoner module within FFIP. FFIP will be reviewed briefly in this paper, followed by the details of a methodology to build the PHM architecture through a StateCharts-based Function Failure Logic reasoner. Finally, the proposed process will be applied to a Liquid Fueled Rocket Engine (LFRE) model to illustrate the potential benefits.

1.1 FFIP for PHM Design

The Function Failure Identification Propagation (FFIP) framework (Kurtoglu and Tumer, 2008a; 2008b; Jensen *et al.*, 2009) 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. The three major modules used in FFIP analysis are the system models based on graphical representations , the behavioral simulator based on and qualitative physics, and the function-failure logic (FFL) reasoner. The graphical system modeling includes the functional model (FM), which 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 as a configuration flow graph (CFG). The CFG is linked directly to the FM 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 FM. These models are then used to build the Function Failure Logic (FFL) reasoner. The FFL resoner uses logic to map component simulation input and outputs to function health or operative state. 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:

$$X(t) = F(\mathbf{c}(t), \mathbf{v}(t)) \tag{1}$$

where $\mathbf{c}(t) = (c_1, c_2, \cdots, c_n)$ is a vector of discrete component modes and $\mathbf{v}(t) = (v_1, v_2, \cdots, v_k)$ 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. As an example, a valve can have liquid flow levels of $\{zero, low, nominal, high\}$ and a sensor may have values of $\{nosignal, high, nominal, low, \}$ 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 to form a system simulation. Failed states are entered into the model, and the finite state computations propagate the failures along the EMS flows to their ultimate ends, which may or may not result in changes in overall system functionality.

FFIP was originally developed to analyze the failure potential of complex systems during the early (highly qualitative) stages of design. It was argued in a prior paper (Kramer and Tumer, 2009) that FFIP has great promise in designing systems with a PHM capability. In that prior paper, FFIP was augmented in three important ways to enable PHM development.

The first augmentation considers catastrophic failures such as violent explosions, which can propagate across functional flows. A nomenclature was added to the FM 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. Ongoing work also considers the state of the flow to capture such explosions (Jensen *et al.*, 2009).

The second augmentation introduces 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 model is critical to assessing the ability of PHM to respond to a given failure. Failures that propagate slowly are failures that PHM can help mitigate or correct. Failures that propagate instantly may not provide the response time a PHM system might require to correct or mitigate the problem, and thus possibly requiring other means of addressing such failures. Nomenclatures were added to the FMs and CFGs capturing these estimations so they can be incorporated in the finite state machine simulation.

The third augmentation 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 understanding the PHM architecture details needed for that particular complex system application.

1.2 Contributions

This paper focuses on applying embedded system development methodologies based on the concept of StateCharts to the construction of the FFL reasoner in a manner that graphically exposes the FFL reasoner and PHM response logic.

In previous work, the CFG and FM system representations only addressed preliminary sensor placement (Kramer and Tumer, 2009). The FFL reasoner relates sensor signals to the PHM response logic through the same hidden finite state machine simulation as above. Application of embedded system design methodologies proposed and conceptually demonstrated in this paper aims to more clearly and visually demonstrate the hierarchy between sensor signals, component states, failure states, failure propagations, and PHM response.

2 RELATED RESEARCH

2.1 Embedded System Design

In this paper, we focus on the software development needs for PHM architectures. PHM, by definition, is an embedded system. In many modern human artifacts, PHM is a system that collects, controls, responds to 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 a liquid-fuel rocket engine (LFRE) example.

Embedded system design is a subfield 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, where the software code is developed in conjunction 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 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 (Zave, 1982; Ernst, 1998; Gajski, 1994).

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 (Harel, 2001). 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 (Gajski, 2003). Based on the preliminary level of data used in FFIP, several of the advanced capabilities of SDL would not be effective. SDL demonstrates the most usefulness when addressing system efficiency through specific circuits applications where the embedded system is a large portion of the overall system architecture. For the applications of interest for PHM integration (e.g., LFRE), 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 StateCharts 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 provide the advantage of demonstrating logic unequivocally and completely, but does so at the expense of simplicity (Peterson, 1981). Typical problems cited when using Petri nets have been state explosion as the system increases in complexity and the inability to represent system hierarchy in a clear fashion. Though future versions of

Petri nets have resolved these problems, they have not been adopted by the software community, and hence are not considered in this research.

2.2 StateCharts

In this research, StateCharts are explored as a means to move the failure propagation analysis capabilities provided by FFIP towards a hierarchical PHM design framework. StateCharts is a common early stage language used in embedded systems design. StateCharts have been used as the foundation for other system development methodologies, such as SpecC and SystemC, both of which are rapidly evolving StateCharts based embedded system and software development processes (Fujita and Hakamura, 2001; IEEE, 2006). Although an industry standard has not emerged from the current fray, most of these methodologies utilize the fundamental strengths of StateCharts 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 (Harel, 2001). Figure 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.

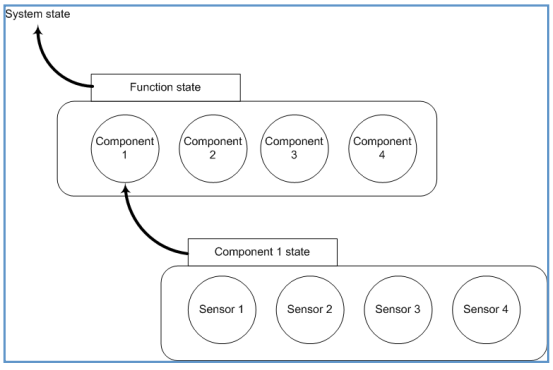

Figure 1: StateCharts Hierarchy.

In a PHM system, the sensors are the tip of the spear for the subsystem. As such, the sensors are the lowest level the StateCharts would reach in this architecture. Each component state will be a function of signals gathered in the system. The sensors are the means by which information is collected and relayed in the PHM system. The mapping between sensor readings and component states is critical for a PHM system.

3 STATECHARTS BASED REASONER

Demonstrated in Figure 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 (Kurtoglu and Tumer, 2008a). For two types of components, a pipe and a valve, the configuration and functional flow graphs are shown. Below these graphical representations, the FFL logic that corresponds to those components is written in common if-then-else 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 StateChart, which contains 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 condition that leads to the state corresponding to that logical flow. Each signal input possibility points to either the next piece of information the reasoner needs or to a 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 StateChart, is reached. From that finite state the arrows then lead up the hierarchy to the function above that particular StateChart. That function (not shown for space considerations) will have a separate StateChart 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 StateCharts 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 may change. This 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

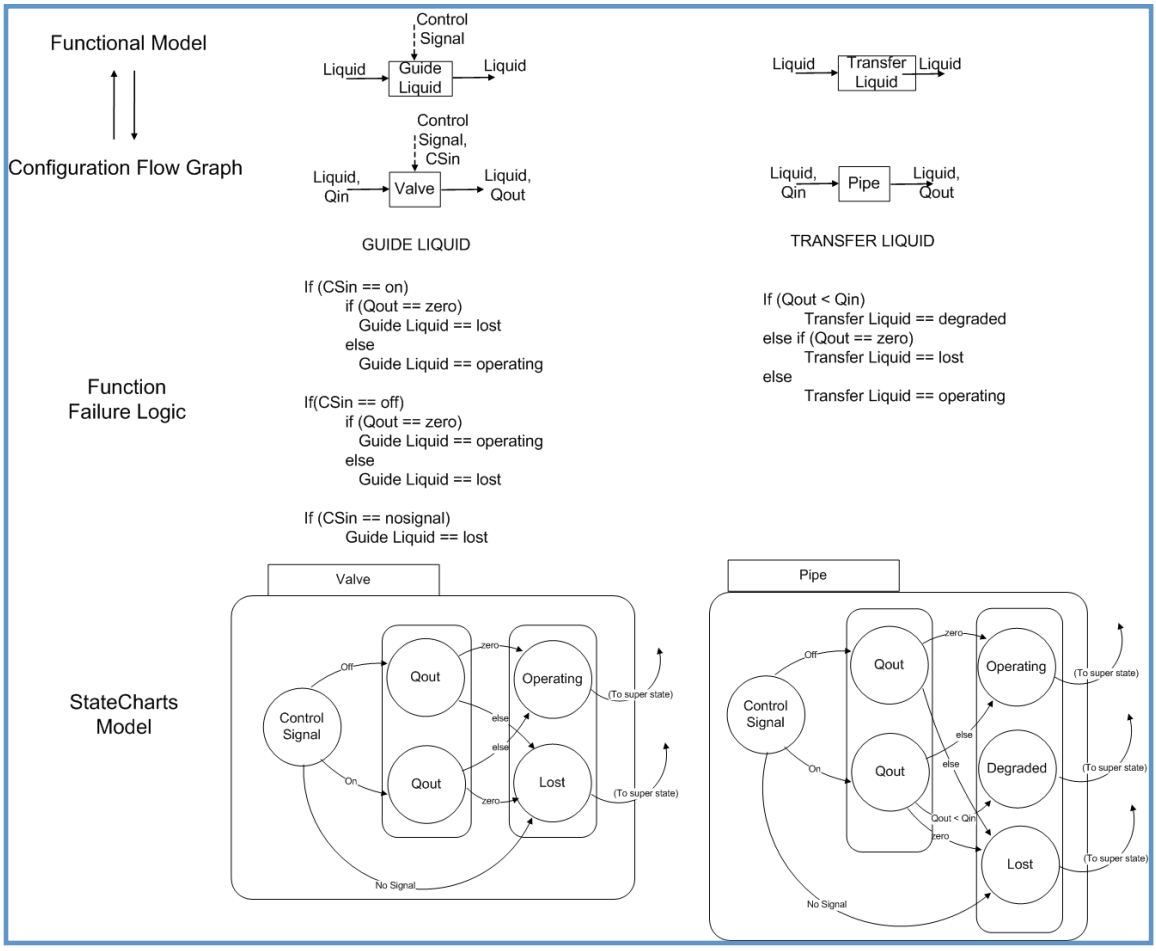

Figure 2: FFIP to StateCharts Mapping Applied to Component Logic.

individual components do not determine the state of the overall system functions, but input data to a higher level StateChart that determines the state of the higher level functions.

4 APPLICATION TO PHM DEVELOPMENT

The existing FFIP framework is briefly explained using a moderately complex Liquid Fuel Rocket Engine (LFRE) architecture as a working example, shown in Figure 3 (Wu, 2005; Sutton, 1986). 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 the 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 of the 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 (Sutton, 1986).

The configurational flow graph (CFG), shown in Figure 4, lists the components and the flows between them. The configurational flow graph maps directly to the functional model (FM), shown in Figure 5, 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 are 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, 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 6 illustrates the FFIP simulation results of 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 some period of time. This is modeled through the system behavioral simulator. 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.

Figure 7 shows a StateCharts representation to demonstrate that the Guide Liquid function for the fuel flow is a function of eight 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 subfunction 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 StateCharts developed. The Coolant Control Valve StateChart lists the finite states that are input to the StateChart and the finite state outputs, sent up the StateCharts hierarchy as well as the expected control signal and sensor states that are down a level in the hierarchy.

A key piece of information drawn from this StateCharts-based representation is an early assessment of sensor placement effectiveness. As each function of the system has a specific StateCharts 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 StateCharts hierarchy to a sensor fingerprint for the various finite states of that particular function. In such a case, PHM designers can see whether 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 StateChart, one can see that the failed states also have numbers next to them. These demonstrate the Time to Failure Propagation discussed in earlier work (Kramer and Tumer, 2009). Again, these are order of magnitude estimates as to how quickly a failure propagates downstream from where it happens. (1 indicates slow prop-

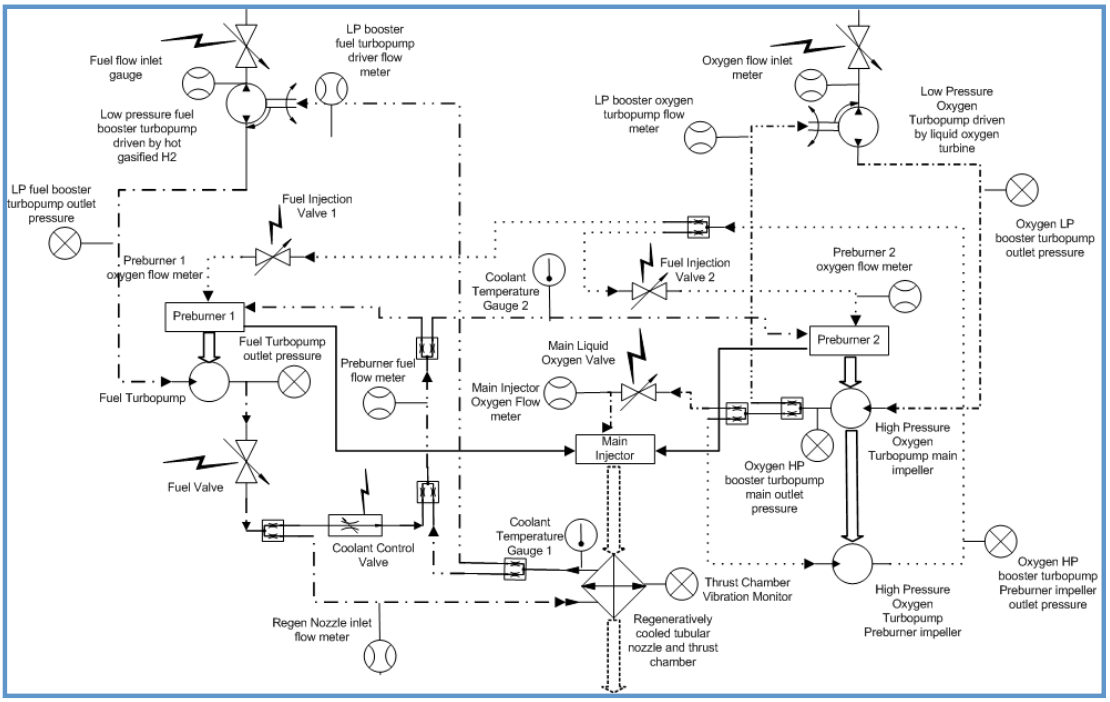

Figure 3: LFRE Schematic.

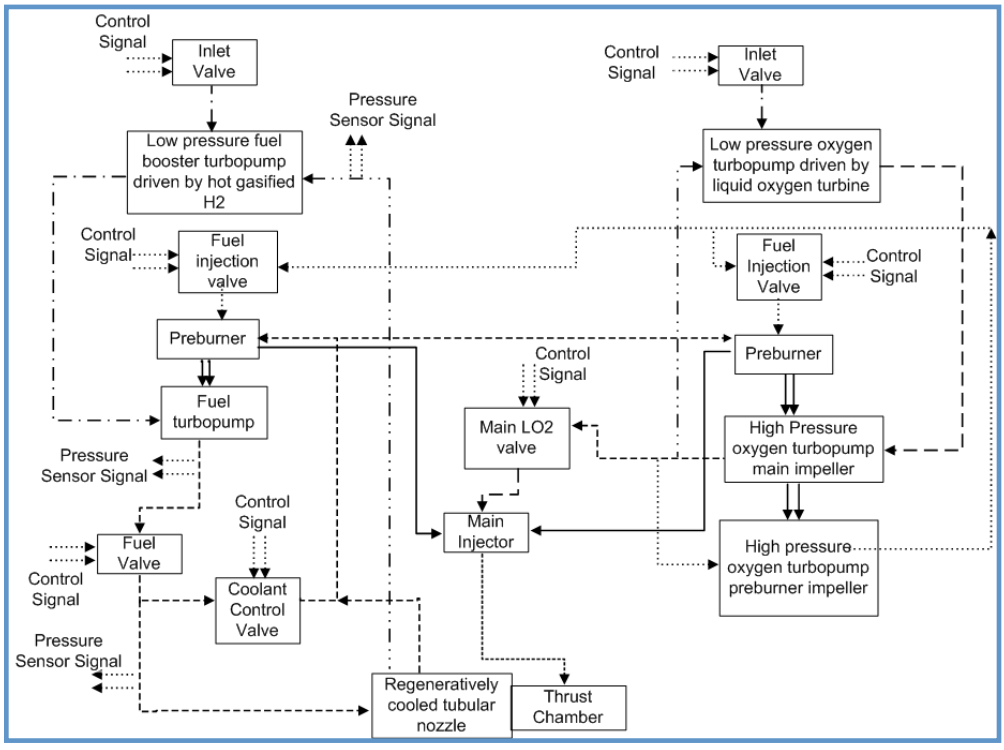

Figure 4: LFRE Configurational Flow Graph (CFG).

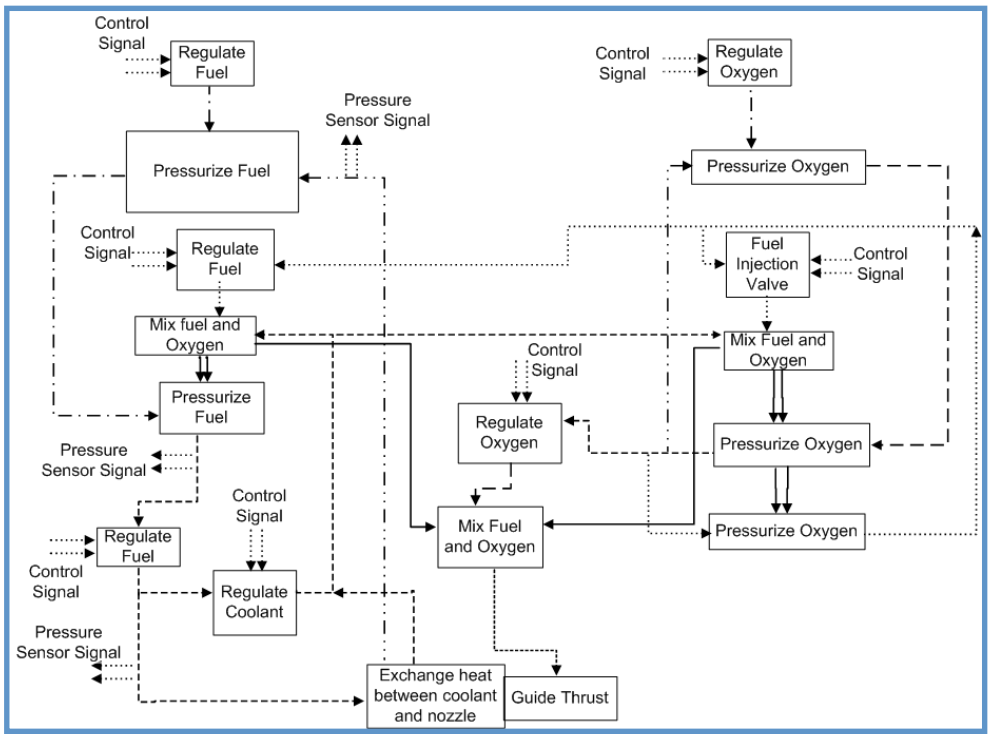

Figure 5: LFRE Functional Model (FM).

agation, 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 a 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 StateCharts 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.

5 DESIGNING PHM SYSTEM RESPONSE

With the system modeled as a finite state machine that builds the Function Failure Logic (FFL), engineers then 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, subfunctions, individual components, and the sensors with which the PHM sees the system. This is important when building the FFL reasoner for 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 subsystem StateCharts will exist next to the system simulation hierarchy.

When a failure is noted in the existing StateCharts architecture and a PHM response is identified, the StateChart for that component will also send a message over to the PHM response hierarchy, as shown in Figure 8. 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 (Harel *et al.*, 1990). This logic should be separate from the finite state logic already built into the system so as to continue to analyze the system logic

	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
Regenerati vely 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 combusti on 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**		
Combusti on	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 Functi ons					
Import Liquid (inlet pipe)		operati ng	operati ng	operati ng	operati ng
Guide Liquid (LP)		operati ng	operati ng	operati ng	operati ng
Guide Liquid (Turbopump)		operati ng	operati ng	operati ng	operati ng
Guide Liquid (Fuel Valve)		operati ng	operati ng	operati ng	operati ng
Guide Liquid (Coolant control valve)		**lost**			
Process signal (Coolant valve controller)		operati ng	operati ng	operati ng	operati ng
Guide liquid, heat (regen nozzle)		operati ng	**high**		
Guide liquid, heat (preburner)		operati ng	operati ng		**high**
Guide liquid, heat (LP turbopump)		operati ng	operati ng	operati ng	operati ng
Guide liquid, (Preburner)		operati ng	operati ng	operati ng	operati ng
Guide liquid, thrust (Main Combusti on Chamber)		operati ng	operati ng	operati ng	operati ng
Measure fl ow		operati ng	operati ng	operati ng	operati ng
Measure vibrati on		operati ng	operati ng	operati ng	operati ng

Figure 6: Simulation: FFL Logic Applied to LFRE Coolant Valve Failure.

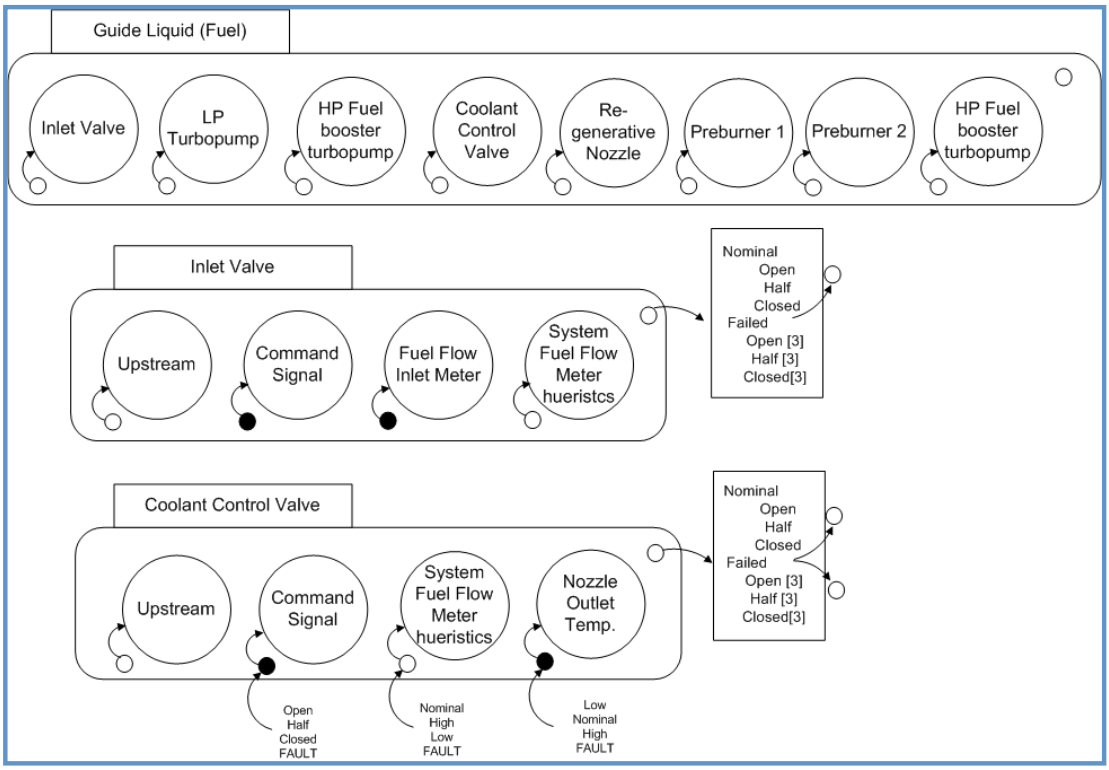

Figure 7: Partial StateCharts Representation of ”Guide Liquid” Functional Hierarchy.

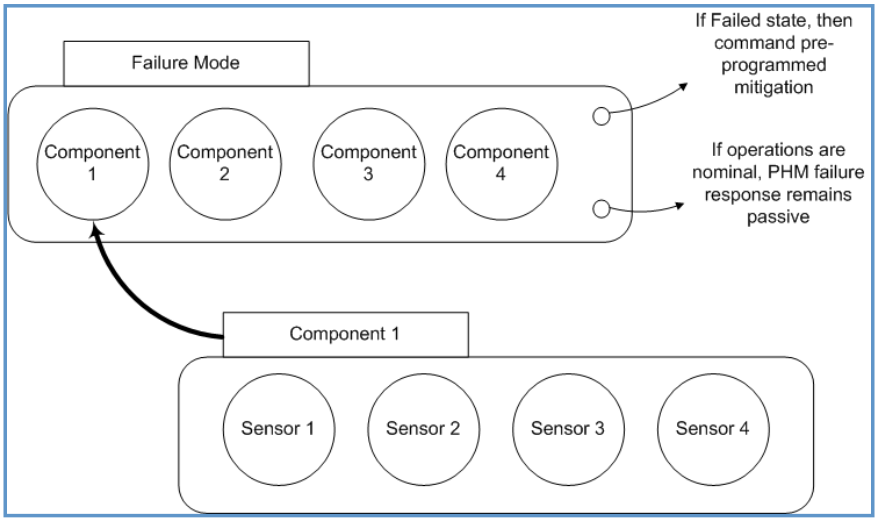

Figure 8: StateChart Representation of PHM System Computational Architecture.

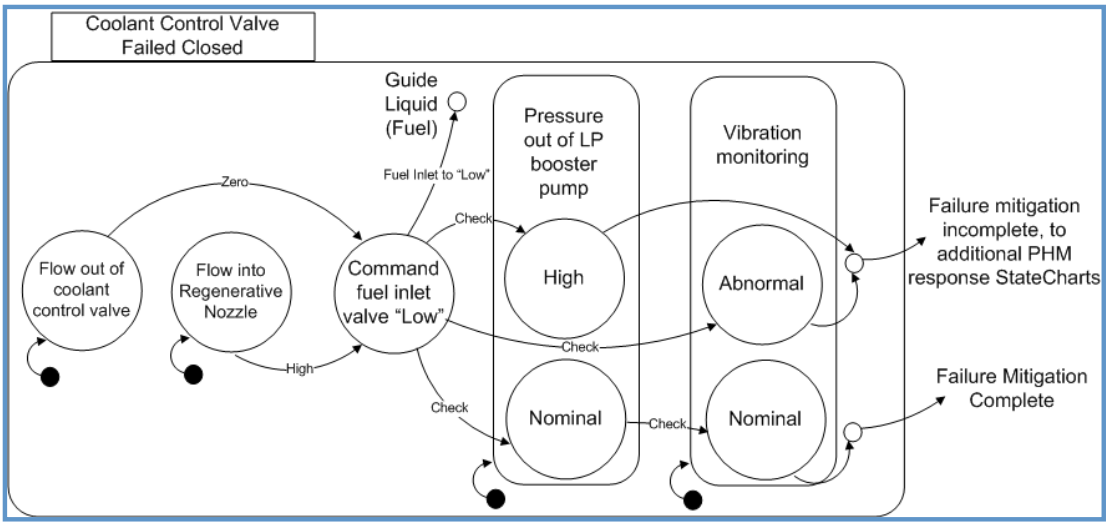

Figure 9: StateChart Applied to PHM Response Logic of Coolant Control Valve Failure Example.

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 brainstorm about potential failures and then ask about what 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 health management systems through defining needs and coding to specific scenarios. PHM goes further by being responsible for detecting the failure and initiating response without operator command. These responses have their programming begin at the conceptual design stage in the PHM response StateCharts.

In previous work to augment FFIP for implementation to LFRE design, PHM response was discussed for handling a slow propagating valve failure (Kramer and Tumer, 2009). 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, the 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, as shown in Figure 9. In this case, we chose to regulate the fuel inlet valve. Although more energy is being carried to the fuel booster turbopump, the fuel flow is regulated to nominal levels with another valve. This response is fed into the upper levels of the StateCharts hierarchy in the form of a command signal changing the state of a component. If the StateCharts are programmed correctly, this change of state propagates through the StateCharts hierarchy resulting in nominal fuel mixtures and thus nominal vibrations.

6 RELIABILITY CONSIDERATIONS

6.1 StateCharts for Sensor Reliability

Knowing the fingerprint of each StateChart can allow for assessment of components that may not have direct sensor data. By using data from elsewhere in the sys-

tem, understanding the hierarchical 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 designers about the efficiency of the overall system design.

StateCharts modular programming construct can address the development of such subroutines 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 StateCharts paradigm to the FFIP process addresses these needs earlier in the design process.

Sensor reliability can also begin to be addressed at this stage. 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 StateChart, 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 subsystem 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 StateCharts 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 StateCharts 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 StateCharts based architecture.

6.2 StateCharts for Software Reliability

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 (Tayler and Hoek, 2007; Caporuscio *et al.*, 2007; Lyu, 2007; Gallardo *et al.*, 2006; McKelvin *et al.*, 2005; Graham, 2005). Some of these processes are modeled on and in many ways emulate processes applied to mechanical systems. Regardless of which process is used, this StateCharts 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.

7 CONCLUSIONS AND FUTURE WORK

This work presents an initial step in using the Function Failure Identification Propagation (FFIP) framework as a tool for PHM design. Through the addition of embedded systems design fundamentals, specifically, a StateCharts representation, 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 cheapest and easiest.

Although this research has focused on PHM design based upon a failure analysis methodology, the rapidly evolving field of unified modeling languages, such as SysML, SystemC and SpecC, 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.

ACKNOWLEDGMENT

This research is supported by the Air Force Office of Scientific Research under grant number AFOSR FA9550-08-1-0158. Any opinions or findings of this work are the responsibility of the authors, and do not necessarily reflect the views of the sponsors.

REFERENCES

(Caporuscio *et al.*, 2007) N. Caporuscio, N. Georgantas, and V. Issarny. A perspective of the future of middleware-based software engineering. *IEEE Future of Software Engineering*, 2007.

(Duncavage *et al.*, 2006) D. Duncavage, F. Figueroa, R. Holland, and C. Schamalzel. Integrated system health management (ishm): Systematic capability implementation. In *IEEE Sensors Applications Symposium*, 2006.

(Edwards *et al.*, 1997) S. Edwards, L. Lavango, E. Lee, and A. L. Langiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. In *Proceedings of the IEEE*, volume 85, 1997.

(Ernst, 1998) R. Ernst. Codesign of embedded systems: Status and trends. *IEEE Design and Test of Computers*, 1998.

(Fujita and Hakamura, 2001) M. Fujita and H. Hakamura. The standard specc language. In *Proceedings of the ISSS*, 2001.

(Gajski, 1994) D. Gajski. *Specification and Design of Embedded Systems*. Kluwer Academic Publishers, 1994.

(Gajski, 2003) D. Gajski. *Embedded System Design*. Kluwer Academic Publishers, Boston, MA, 2003.

(Gallardo *et al.*, 2006) M. Gallardo, J. Martinez, P. Merino, and E. Pimentel. On the evolution of reliability methods for critical software. *Transactions of the Society for Process and Design Science*, 10(4), 2006.

(Graham, 2005) J. H. Graham. Fmeca control for software development. In *Proceedings of the 29th Annual International Computer Software and Applications Conference*, 2005.

(Harel *et al.*, 1990) D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, 1990.

(Harel, 2001) D. Harel. Statecharts: A visual formalism for computer systems. *Science of Computer Programming*, 8:231–274, 2001.

(Hoyle *et al.*, 2009) C. Hoyle, I. Y. Tumer, and and W. Chen A. F. Mehr. Health Management Allocation During Conceptual System Design. *Journal of Computing and Information Science in Engineering*, 9(2), 2009.

(Hutcheson and Tumer, 2005a) R. Hutcheson and I. Y. Tumer. Function-based Co-design Paradigm for Robust Health Management. In *Proceedings of the 5th International Workshop on Structural Health Monitoring*, 2005.

(Hutcheson and Tumer, 2005b) R. Hutcheson and I. Y. Tumer. Function-based design of a spacecraft power subsystem diagnostics testbed. In *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, 2005.

(IEEE, 2006) IEEE. *IEEE Standards SystemC*. Prentice Hall, 2006.

(Jensen *et al.*, 2008) D. Jensen, I. Y. Tumer, and T. Kurtoglu. Modeling the propagation of failures in software-driven hardware systems to enable risk-informed design. In *Proceedings of the ASME International Mechanical Engineering Congress and Exposition*, 2008.

(Jensen *et al.*, 2009) D. Jensen, I. Y. Tumer, and T. Kurtoglu. Flow State Logic (FSL) for analysis of failure propagation in early design. In *Proceedings of the ASME Design Engineering Technical Conferences; International Design Theory and Methodology Conference*, 2009.

(Kramer and Tumer, 2009) S. Kramer and I. Y. Tumer. A framework for early assessment of failures during the design of PHM systems. In *Proceedings of the ASME Design Engineering Technical Conferences; Computers and Information in Engineering Conference*, 2009.

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

(Kurtoglu and Tumer, 2008b) T. Kurtoglu and I. Y. Tumer. A risk-informed decision making methodology for evaluating failure impact of early system

designs. In *Proceedings of the ASME Design Engineering Technical Conferences; International Design Theory and Methodology Conference*, 2008.

(Kurtoglu *et al.*, 2008) T. Kurtoglu, S. Johnson, E. Barszcz, J. Johnson, and P. Robinson. Integrating system health management into early design of aerospace systems using functional fault analysis. In *Proc. of the International Conference on Prognostics and Heath Management, PHM08*, 2008.

(Leao *et al.*, 2008) P. Leao, Bruno, T. Yoneyama, G.C. Rocha, and K.T. Fitzgibbon. Prognostics performance metrics and their relation to requirements, design, verification, and cost-benefit. In *Proc. of the International Conference on Prognostics and Heath Management, PHM08*, 2008.

(Lyu, 2007) M.R. Lyu. Software reliability engineering: A roadmap. *IEEE Future of Software Engineering*, 2007.

(M. *et al.*, 2005) Roemer M., Byington C., Kacprzynski G., and Vachtsevanos G. An overview of selected prognostic technologies with reference to an integrated phm architecture. In *Proc. of the First Intl. Forum on Integrated System Health Engineering and Management Conference*, 2005.

(McKelvin *et al.*, 2005) M. L. McKelvin, G. Eirea, C. Pinello, S. Kanajan, and A. L. Langiovanni-Vincentelli. A formal approach to fault tree synthesis for the analysis of distributed fault tolerant systems. In *Proceedings of the EMSOFT'05*, 2005.

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

(Sutton, 1986) J.P. Sutton. *Rocket Propulsion Elements: An Introduction to the Engineering of Rockets*. John Wiley and Sons, New York, 1986.

(Tayler and Hoek, 2007) R.N. Tayler and A. Hoek. Software design and architecture: The once and future focus of software engineering. *IEEE Future of Software Engineering*, 2007.

(Tumer, 2005) I. Y. Tumer. Towards ISHM Co-Design: Methods and practices for fault avoidance and management during early phase design. In *Proceedings of the 1st Integrated Systems Health Engineering and Management Forum*, 2005.

(Wu, 2005) G. Wu. Liquid-propellant rocket engines health-monitoringa survey. *Acata Astronautica*, 56(3):347–356, 2005.

(Zave, 1982) P. Zave. An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering*, 1982.

Scott Kramer has been an active duty member of the U.S. Coast Guard since the summer of 1999. Upon graduating high school in Ft. Mohave, AZ, Scott attended the U.S. Coast Guard Academy in New London, CT graduating with a BS in Naval Architecture/ Marine Engineering in the summer of 2003. Since then, Scott has served as a commissioned officer in various duty stations including the Polar Class icebreaker Polar Sea homeported in Seattle, WA and as a ship superintendent at the U.S. Coast Guard Yard in Baltimore, MD. During his Coast Guard career, Scott has sailed some of the most unique ships in the US Government fleet to some of the most remote locations in the world, faced down real life engineering problems in remote locations, managed multi-million dollar ship repair operations, and acted as a liaison with foreign services. While attending graduate school at Oregon State University, Scott leveraged some of this real life experience to study risk mitigation and analysis methodologies for early stage complex system design. Having received his Masters degree from OSU in Mechanical Engineering in the Spring of 2009, Scott is continuing his career as a Coast Guard officer.

Dr. Irem Tumer is an Associate Professor at Oregon State University. Her extensive experience at NASA and in the Engineering Design community has led to over 20 journal publications and over 80 refereed conference publications, focusing on risk and failure analysis and engineering design theory and methodology. Prior to accepting a faculty position at OSU, Dr. Tumer led the Complex Systems Design and Engineering group in the Intelligent Systems Division at NASA Ames Research Center, where she worked from 1998 through 2006 as Research Scientist, Group Lead, and Program Manager. She has been extensively funded through various NASA programs while leading the Complex Systems Design group during her time at NASA Ames Research Center between 1998 and 2006, and through NSF and AFOSR since moving to Oregon State University in 2006. Dr. Tumer has been Conference Chair for ASMEs Design for Manufacturing and the Lifecycle conference in 2000, and Technical Program Chair for IEEE Reliability Societys Prognostics and Health Management Conference in 2008, and Symposium Organizer and Chair for Integrated Systems Engineering at ASME's Computers in Engineering Conference in 2008 and 2009. She received her Ph.D. in Mechanical Engineering from The University of Texas at Austin in 1998.