Toward a Model-Based Approach to Flight System Fault Protection

John Day
Alex Murray
Peter Meakin
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
John.Day,Alex.Murray,Peter.Meakin@jpl.nasa.gov

Abstract—Fault Protection (FP) is a distinct and separate systems engineering sub-discipline that is concerned with the off-nominal behavior of a system. Flight system fault protection is an important part of the overall flight system systems engineering effort, with its own products and processes. As with other aspects of systems engineering, the FP domain is highly amenable to expression and management in models. However, while there are standards and guidelines for performing FP-related analyses, there are not standards or guidelines for formally relating the FP analyses to each other or to the system hardware and software design. As a result, the material generated for these analyses are effectively creating separate models that are only loosely-related to the system being designed. Development of approaches that enable modeling of FP concerns in the same model as the system hardware and software design enables establishment of formal relationships that has great potential for improving the efficiency, correctness, and verification of the implementation of flight system FP. This paper begins with an overview of the FP domain, and then continues with a presentation of a SysML/UML model of the FP domain and the particular analyses that it contains, by way of showing a potential model-based approach to flight system fault protection, and an exposition of the use of the FP models in FSW engineering. The analyses are small examples, inspired by current real-project examples of FP analyses.

TABLE OF CONTENTS

1  INTRODUCTION .......................... 1
2  FAULT PROTECTION DOMAIN DESCRIPTION ... 1
3  THE SYSTEM USED FOR OUR ANALYSIS ........ 2
4  FAULT PROTECTION ARTIFACTS AS MODELS .. 4
5  FUTURE WORK ............................ 16
6  SUMMARY .................................. 17
7  ACKNOWLEDGMENTS ....................... 17
8  REFERENCES ............................. 17
9  BIOGRAPHY ............................... 17

1. INTRODUCTION

Development of modeling approaches for flight system FP first requires reviewing the analyses and products typically used in flight system FP development and how they are related. Typical analyses and products developed as part of a flight systems fault protection development effort include: 1) a Failure Mode, Effects, and Criticality Analysis (FMECA), whose objective is to comprehensively identify all of the ways in which system components can fail, and identify the consequences and severity of each failure mode; 2) a Fault Tree Analysis (FTA), which is a top-down identification of system or subsystem failures, and a decomposition of each failure into a logic tree showing all possible combinations of sub-failures that could have resulted in the top-level failure; 3) a Monitor and Response Dictionary (MRD) that specifies conditions that must be monitored at runtime, and responses at various levels to the occurrence of those failures; and 4) a set of mitigation matrices that bookkeep how all of the failures modes identified in the FMECA are detected and/or corrected at runtime, or shown through testing or analysis to be unlikely to occur.

In and of itself, this FP domain is highly amenable to expression and management in models. In addition, both the FMECA and the FTA rely heavily on system descriptions such as block diagrams to define the components and subsystems to which they refer. Thus, the systems engineering activities of the FP domain could be well served with the use of integrated systems models, shared with the larger systems engineering team, upon which to base their analyses.

Though there are industrial and government standards for performing FMECA's, and others for FTAs, there is not a unified standard that links the two, or that relates either the FMECA or the FTA to the MRD, or to other general system models. Current practice tends to rely on spreadsheets to develop and contain all of the major products of FP engineering: the FMECA, the FTA, the MRD, and the mitigation matrices. This approach is lacking for several reasons: since the FMECA is a model of the flight system, it effectively creates a second baseline of the system design, separate from that used by the systems engineering team at large; it provides no or limited ways to describe behaviors, such as failure modes or failure effects; and it provides only brittle and limited support for managing relationships among elements, e.g., from a failure mode to a mitigating fault response.

Flight system FP is also tightly coupled with flight software (FSW), because of the need to exhaustively specify the runtime detection and mitigation of failures. This is another advantage of a model-based approach for FP: FSW engineering is highly amenable to model-based approaches, and the sharing of models between the two domains has great potential for improving the efficiency, correctness, and verification of the implementation of runtime FP.

2. FAULT PROTECTION DOMAIN

Fault protection is the aspect of systems engineering that is responsible for the design of the off-nominal behavior of a system. As used and applied at JPL, Fault Protection
(FP) is both a specific systems engineering discipline (similar to End-to-End Information System (EElS) engineering or mission planning), and the functions and elements of a system that address off-nominal behavior. While Fault Protection (FP) is the historical term for this field at JPL, it is also known by a variety of other names, including Vehicle Health Monitoring, Vehicle Health Management, Integrated Diagnostics, Prognostics and Health Management, Fault Detection, Isolation, and Recovery (FDIR), and Redundancy Management (RM). Recently within NASA, Fault Management (FM) is gaining popularity as the primary term to refer to the field.

Fault protection systems engineers develop requirements and architecture, perform trades, analyze and characterize the system, verify and validate the system, and operate the system. This is done in concert with other systems and subsystem engineers, reliability engineers, system architects, test engineers and operations engineers. The FP engineer makes use of various techniques to determine the failure space, assess threats against the mission objectives, and define functionality to mitigate threats with some desired coverage over the failure space (both identified and unidentified). Engineers working NASA robotic and human spaceflight programs have developed a variety of methods for fault mitigation, fault analysis, and contingency operations over the course of several decades. These include fault tree and failure modes and effects analyses (FMEAs), redundancy management, fault detection, isolation and recovery (FDIR), vehicle health management, troubleshooting, fault tolerance, integrated diagnostics, and contingency planning, among others. The variety of terms and methods is a symptom of the integrated diagnostics, and contingency planning, among other things. The design and performance objectives. Many different architectures are possible for deploying FP functionality, but in this paper we focus on the typical monitor/response pattern. The connection between the failure space and the FP monitors and responses is not always formally captured, but when it is, it is typically captured in a mitigation matrix. Capturing monitors/responses, FMECA results and FTA results in a SysML model provides a way to represent these elements and their relationships in a formal and consistent way.

FP functions to detect, diagnose, decide and respond to off-nominal conditions are deployed at various levels, with system, subsystem, and component level detections and mitigations. Determination of the fault protection functions to apply and where to apply them in done by both top-down and bottoms-up analytical approaches. For each system function to be protected, failures can either be prevented or mitigated. FP is deployed when failures are to be operationally mitigated, but is not necessary when the function is preserved by prevention. If the function cannot be preserved, then alternate goals may be selected.

A key factor to determine where in the function tree one must deploy FM mechanisms is the race condition of failure effect propagation times versus failure mitigations. For a FM mechanism to be effective, the mitigations must "win the race" against the corresponding failure effects. If the mitigation loses the race, then the failure mitigations must be implemented at a location in the design that is closer to the originating failure mode, thus reducing the detection time, and/or with a response mechanism that is faster than some higher-level mechanism. Analysis of these race conditions implies the need to model failure effect propagation and identify all FM control loops.

Assessment of the FM design also entails bottom-up analysis of the failure effects that result from the multitude of system failure modes. Bottom-up analyses draw from failure modes and effects analyses (FMEAs), which provide information regarding the ways in which the system’s components fail, and the effects that result. These effects are the behaviors that FP detects and to which it responds. Precise representations of these failure effects are required, so as to better understand and analyze them.

In general, FMEAs are quite accurate in their description of the failure modes (mechanisms) of the component, and in the immediate failure effect or failure symptom caused locally at the interface of that component to the rest of the system. There are usually further fields that describe downstream effects, but these often become progressively more inaccurate the further away the effect is from the originating component. This is because the methods used to determine the downstream effects can vary depending on the FMEA analyst or designer that contributes to the writing the English-language text descriptors.

The FM design process interoperates with existing systems engineering (nominal) design processes and with failure analysis (off-nominal) design processes. For systems engineering, the primary links will be to the function tree representation of system functional decomposition and to the nominal event sequences that are tied to concepts of operation. For failure analysis, the primary links are to bottom up FMEAs and top down fault tree analyses and representations. The result of the analysis and design work is the set of fault protection monitors and responses that have been developed to adequate cover the failure space (within the defined risk and performance objectives). Many different architectures are possible for deploying FP functionality, but in this paper we focus on the typical monitor/response pattern. The connection between the failure space and the FM monitors and responses is not always formally captured, but when it is, it is typically captured in a mitigation matrix. Capturing monitors/responses, FMECA results and FTA results in a SysML model provides a way to represent these elements and their relationships in a formal and consistent way.

FP has been successfully applied to many flight systems, the approaches are generally ad hoc and result in gaps and inefficiencies in the overall FP design (and is especially problematic for system-of-systems programs). These methods are also unable to answer, or only partially address important questions relating to characteristics such as the completeness and effectiveness of the FP design. There is significant benefit to be gleaned from applying greater rigor and a more systematic approach to FP development, and that the burgeoning field of Model-Based Systems Engineering can provide useful techniques and tools to help us in this endeavour.

3. The System Used For Our Analysis

This section first describes our hypothetical flight system. Our model is loosely based on a typical Earth orbiter flight system design, and presents a high-level sketch of a flight system, with more detail filled in only where necessary to illustrate our approach to modeling flight system fault protection. It bears emphasizing here that we assume that the sort of system model we are showing here would be produced by the larger flight system systems engineering team, and FP engineers would add attributes to this model as necessary. Thus, this model would be a well-reviewed product of a collaborative process. Indeed, one of the key advantages of this approach is reducing the duplication of effort that is typical in a more traditional process, in which FP engineers essentially make their own system model.
Figure 1. Top-level flight system decomposition

Figure 2. The Structures and mechanisms subsystem components
The System Model

Figure 1 shows a high-level decomposition of the components of the flight system. This structural decomposition view is just one view of a model made for general systems engineering usage, but it is the view most relevant to this paper, because, in our approach, failure modes are described as attributes of the physical (or, in the case of software, logical but none the less concrete) components of the system.

The flight system we model is composed of the key, typical subsystems. We show a further level of decomposition for a couple of the subsystems. Shown in Figure 2 is a partial decomposition of some of the mechanical and structural components of the system, with the Launch Vehicle Interface component decomposed into some of its key parts. The lowest level of this decomposition involves mechanical assemblies and components, given generic names in our figure, and these components will be involved in the discussion of failure modes and fault trees, below.

Turning to the Guidance, Navigation and Control component, we present a decomposition of it in Figure 3. The software part of GNC is depicted as shared between this and the FSW component. The decomposition of the GNC part of FSW is done functionally, e.g. GncDevManagement, and represented in the model as behaviors, and represented here as boxes with the activity stereotype (a SysML/UML activity is a behavior, represented in detailed views, not shown here, with flow chart-like notation). We chose a component involving software to illustrate its treatment in the FMECA, which we'll describe below.

4. Fault Protection Artifacts As Models

In this section we begin with a discussion of the SysML and UML modeling languages in order to explain how we extend them to express the concepts of the FP domain. We then show examples of using our extensions to express the FP analyses in our UML/SysML model.

The Profile

First, a little background: the UML is made to be extensible, and so it provides an extension mechanism in the form of a metamodel construct called Profile (SysML extends UML using this profiling construct). A good concise description of the profiling mechanism is given in [5]. UML is based on a metamodel, which, in a nutshell, is a model of how models are made. The metamodel consists of metaclasses, which embody the concept of modeling a particular type of thing.

So for example, our SysML model contains a class called GNC Device 1. This “device” might represent some common GN&C device, let’s say a star tracker. This class is a type that expresses the idea of a set of things that share a set of attributes which all start tracker devices have: tracking performance, size, power usage, electrical connection types, digital communication interface, etc. The very concept of expressing such an idea in a model is embodied in the UML metamodel as the metaclass class.

In the profiling mechanism, a metaclass can be extended: have additional attributes given to it, or additional constraints placed upon it, by defining a special kind of metaclass called a stereotype. A stereotype is attached to one or more metaclasses (those which it extends). In a UML or SysML model, when an instance of a metaclass is created (e.g. defining the
class GNC Device 1 instantiates - creates an instance of - the metaclass class, the stereotype can be applied to that instance. In fact, the SysML metaclass block is actually a stereotype on the UML metaclass class, and that stereotype has been applied to the GNC Device 1 class, as shown by the word <<block>> on the GNC Device 1 shape in Figure 3.

To extend SysML and UML for our purposes, we defined a set of stereotypes. It’s convenient to describe these as grouped into stereotypes that describe things, and those that describe relationships between things. The key stereotypes of the first group are shown in Figure 4. In UML/SysML, classifiers of any kind - blocks, classes, or metaclasses among others - are can be represented as boxes, with selectable levels of detail (e.g. whether to show a class’ attributes or not). The stereotype FailureMode is shown in red (we assign colors to the stereotypes and use the colors consistently when applying them). Stereotypes extend one or more metaclasses, and in this diagram, the extended metaclass is shown in square brackets. The metaclass extended by FailureMode is BehavioredClassifier, which is the base metaclass of all of the behavior kinds in UML/SysML: state machines, activities, sequences, and other kinds of behavioral models. Thus, by choosing BehavioredClassifier as the metaclass for our FailureMode stereotype, we embody a concept of failure modes as behaviors: a given component behaves in certain ways, some of them nominal, some of them not.

The FailureMode stereotype has four attributes (attributes of stereotypes are called tag definitions, or simply tags, in UML/SysML). These identify the likelihood of the occurrence of a failure mode, the severity of the mission impact, as well as a list of SystemModes in which the failure mode presents a risk to the mission.

For example, a spring normally continually applies force in a given direction; this is the nominal behavior. However, the spring can break, and stop applying force. This is another behavior, and it is a failure mode. It could be equally well described in an activity or in a state machine.

A failure mode has one or more causes, which we tag in our model with the Cause stereotype, as well as effects, either local (SubsystemEffect), or system-level: SystemEffect. The metaclass for the Cause and Effect stereotypes is Classifier: in UML this is the abstract metatype of not only regular class-like metaclasses such as class and block, but also of behaviors such as state machines, etc. This allows the application of our Cause, SubsystemEffect and SystemEffect stereotypes to regular classes or blocks, or to behaviors. Modeling effects as behaviors can be advantageous because it allows them to appear in fault trees, as we’ll show later.

A FaultTree maps to the fault tree analysis concept described in the introduction. The metaclass BehavioredClassifier allows us to model fault trees as any kind of UML/SysML behavior. In the examples we give below, we use activities to model them, though a state machine might be just as apt a model for them.

The Monitor is another key stereotype in our FP profile. It embodies the concept of behavior in the system that actively monitors some particular aspect of the system, and may take some action based on the state of the monitored feature. These behaviors are best modeled as UML/SysML behaviors; thus the choice of BehavioredClassifier as the metaclass for the Monitor stereotype. The monitored aspect might be something physical, such as a temperature or voltage, or something logical, such as the error in attitude estimation. Typical constraints and requirements on fault monitors are represented by the following characteristics we have built into the model-a unique ID (for use in commanding and telemetry), a minimum frequency of detection of the monitored feature (e.g. a certain temperature is required to be checked for being in range at least once every 20 seconds), and a specification of whether the monitor, once having detected an symptom, can revert to a state of not having detected it without ground intervention.

Responses, both local and system-level, are also best described as behaviors. Indeed, system responses tend to be complex behaviors that really benefit from being modeled in order to be well understood and specified. System responses consist of an ordered set of sub-behaviors called tiers; this embodies the notion of taking successively more aggressive measures in the face of a fault that has not been repaired. System responses also have a priority, used to decide among competing responses which one to execute.

Verification activities include test scenarios, run against the system, and also analyses, which can be used to verify some functionality that’s difficult to test, or to argue that a failure mode is unlikely enough to not need handling. Examples of these items, including their representations and how they relate to other elements of the model are described later.

There are key relations in the FP domain. The metaclasses upon which the stereotypes shown in Figure 5 are relationships. An Association in UML/SysML is a relation between two classifiers, and tends to be used to describe a close relationship, including a sense of ownership or part-to-whole relationship. A failure mode is an intrinsic property of a component, and so this close association metaclass seems appropriate to model the relationship of a component to its failure mode(s). Thus the failsIn relation, used to mark an association between a component (block) and its failure mode or modes, is based on Association.

The metaclass Dependency represents a looser relationship, more appropriate for expressing the other relations in our domain, such as causedBy - the relation from a Cause to a FailureMode, and produces - the relation from a FailureMode to the effect(s) it produces. The detects relation connects a Monitor to a FailureMode, while tolerates maps a Response (Local or System) to a FailureMode against which the Response protects the system. The tolerates relation is a special kind of the mitigates relation (that’s the meaning of the inheritance relation in the diagram - the lines with hollow arrow head); tolerates means that the failure mode is actually dealt with by the running system, whereas mitigation can also mean a pre-fielding, design-time measure to protect against the effects of a failure mode. The avoids relation, linking an analysis to a failure mode, means that the analysis shows that the failure mode cannot plausibly happen, or the resulting risk if it did happen is acceptable.

The executes relationship maps a Monitor to a SystemResponse. This mapping is typically dependent upon the mode of the system, and that is why the executes stereotype has the two tags of type SystemModes, namely validInModes and notValidInModes - these allow an executes relation to be specified to be valid, or in force, only in certain system modes, or in any modes except for a named set of modes.

With this description of the profile, we proceed to examples of applying it.
Figure 4. The stereotypes that express things in the FP domain

Figure 5. The stereotypes that express relationships in the FP domain
The FMECA

For our first example, we present a small section of the FMECA that handles some of the launch vehicle separation components, as shown in Figure 6. The SepSysComponentFailure mode describes scenarios in which the component breaks or bends during launch, causing a subsystem effect of the spacecraft not being supported by the launch vehicle, and a disastrous system effect of premature separation and loss of vehicle (LOV).

This failure mode could be caused by unexpectedly high loads on the component (in turn probably caused by an analysis or requirements error, not shown), or by workmanship.

The failure modes of the other parts of the launch vehicle separation system are similarly depicted. Since all three of the shown failure modes are disastrous, they are assigned missionImpact 6. These failures can only occur during our hypothetical mission phase 1, as reflected by the missionPhase tag value. Note that causes may be shared among failure modes, as can effects (e.g. LOV: Failure to Separate is produced by both Activation Failure and Mechanical Failure).

This example involves mechanical failures. The FMECA must also treat functional failures, especially in software. The next example, in Figure 7, shows some of these failure modes that can occur with the functional parts of the GN&C software. These modes are rather abstract and non-specific.

The failure mode Some Error might represent a failure on the part of operations or commanding (i.e. giving invalid or flawed commands to the spacecraft), or data transport (corrupted data), and the GN C sequencing function of the software’s failure to catch the errors. This failure mode could cause the local GNC subsystem effect of erroneous commands to GNC devices, or the system effects of anomalous attitude, or worse, loss of vehicle.

Note that the produces relationship from the failure mode to the LOV system effect has a constraint “FM occurs during critical event” attached to it, signifying that this failure mode can only produce this effect if a critical event is in process.

It’s important to mention that the components shown in the FMECA examples, e.g. SepSysComponent or GncSequencing, are the same model elements as shown in previous diagrams, except that in this view they are shown with their failure modes: the same blocks are used in both diagrams. Similarly, some of the same failure modes depicted in these FMECA diagrams will also appear in fault tree diagrams, shown in the next section.

Fault Propagation

We can also use our FMECA model to discuss the propagation of failures. We can map an effect - system or subsystem - to another effect, using a produces relation, though we are not showing that in the examples here. The effects involved in these mappings would not necessarily be in the same subsystem, in fact in many cases they would not be. We could then discover a set of directed graphs in the model in which a failure mode was the starting point, and the chain of produced effects would be a possibly branching, possibly even cyclic, network. This seems an area rich in possibilities for analysis of the model to validate the physical model and the FMECA, and fault trees.

Fault Trees

The top-level fault tree is shown in Figure 8. We represent these fault trees as SysML/UML activities. Activities represent a behavior; there is no concept involved of whether the behavior is expected to actually occur in the fielded system or not. Thus activities can be used to represent scenarios that we hope will never occur.

We could also use state machines to represent fault trees, but activity diagrams, with their flow chart-like style, provide a better mechanism for representing the structure of fault trees. The choice of state machines versus activities to represent fault modes is a matter of style: some people think more naturally in terms of modes of behavior, reacting to events, and state machines naturally capture this style of conception. Activities, on the other hand, lend themselves to a more procedural, step-by-step view of a behavior. One or the other may be a better fit for a particular failure mode. In terms of the modeled relations of a failure mode to other aspects of the FM model, it doesn’t matter: either a state machine or an activity can participate in a fault tree (modeled as an activity).

In the top-level fault tree, the end action is that the system fails, meaning the entire mission is lost. We attempt to list all of the ways in which that could have happened. As the figure shows, we identified six possibilities: one of these six behaviors must have happened if System Fails was reached.

Each of these six requisite behaviors is in turn modeled as an activity, which allows us to decompose the fault tree into more and more detail, at each level of decomposition finding more specific and detailed behaviors leading to the result at that level. The behavior 1.1 Launch or Commissioning Failure is further decomposed in Figure 9, into two possible sub-behaviors: Fail to Establish Orbit and Fail to Get Operational. We give our theoretical system a spinning reflector, and so one way of failing to become operational is to have a Spinup Failure.

The first of these is in turn decomposed into the more detailed behavior shown in Figure 10, and at this level of detail, elements from the FMECA - failure modes and system effects of failure modes, begin to come into play: of the four possible behaviors that lead to failing to establish orbit, three of them are made possible by the occurrence of failure modes, and their produced system effects.

Ideally, every fault mode would appear in some part of the fault tree, and every part of the fault tree could be decomposed down to identified failure modes. With the representations that we have defined here, it is then possible to query the model to check that all failure modes for a given effect are represented in the fault tree. This is one way in which the use of models can ensure completeness between these analyses.

Another section of the fault tree, the decomposition of Fail to Get Operational, appears in Figure 11. In our theoretical mission design, the reflector has to spin at a given rate in order to obtain science data, and GNC would be required to maintain vehicle stability during spin-up and also when the science spin rate is achieved.

The failure to spin up the reflector (and maintain control of
Figure 6. The failure modes of some parts of the launch vehicle interface

Figure 7. GN&C failure modes
Figure 8. The highest levels of a fault tree

Figure 9. Launch or commissioning fault scenarios

the spacecraft) is depicted in Figure 12. In this scenario, GN&C software errors and ground commanding errors play a role. Again, failure modes and effects identified in the FMECA come into play.

Fault Monitors and their Mappings

Fault monitors detect errors caused by faults at runtime. Fault monitors are related to the failure modes that they can detect. They are also related to responses that they may cause to be executed. Fault responses are related to the failure modes that they protect against or handle. Figure 13 shows a notional set of GN&C fault monitors. The diagram shows, for example, that the monitor of Excess Attitude Estimation Error can detect the failure mode Incorrect SW Behavior, which was identified in the FMECA.

That monitor is also shown to execute the system fault response SYS_MODE_1 (which causes entry into a defined system mode of Safe Mode with Reaction Wheels). However, note that the constraints shown indicate that this monitor can only cause SYS_MODE_1 to be executed in system modes other than SAFE_RWA or PREMISSION_PHASE_1.
The monitor *Device Overflow* monitors a mode in which some GNC device's data is collected at higher-than-normal resolution - a not atypical feature of flight systems. If the GNC software did not compute timings properly for managing the data (the *Incorrect Phasing* failure mode), then this mode could cause a buffer overflow. There is a local response to reset the device. The local response is represented to mitigate the failure mode.

**System Fault Responses**

The definitions of system fault responses, and also the specification of the management of the execution of responses, including the logic of responding to monitors reporting failures, are key pieces of a the fault protection architecture of a system.

Figure 14 describes some of these key aspects of the FP architecture, showing a concept of a FP engine that manages reacting to symptom reports from monitors, and executing system responses. Recall that system responses consist of an ordered set of one or more sub-behaviors called tiers, and that executing a response means executing each of its tiers, in order. The association from the engine to the system responses, called *Executes A Tier Of*, is marked with a constraint called *eligibleForExecution*, which would be defined in great detail in any FP architecture. A typical constraint that might be part of the definition of eligibleForExecution would be whether or not the mapping from monitor to response is applicable in the current system mode.

Establishing these key properties of fault management is essential. Models can be used to simulate the response behaviors and their interactions, leading to an improved understanding of the implications of these architectural choices.
Verification and Validation

The fault protection aspects of the system must be verified and validated just as any other aspects of the design must be. The goals of FP validation include: 1) establishing the completeness of the FMECA — have all plausible failure modes been identified?, 2) validating the completeness of the fault tree analyses — have all paths to failure scenarios been identified?, 3) establishing the completeness of mitigation of known failure modes.

The goals of verification in FP include: 1) verifying the detection logic of all monitors, 2) verifying the execution relationships between all monitors and responses, 3) verifying the correctness of the logic of all responses.

The key elements of a V&V effort are test cases and analyses, and their relationships to failures modes, fault monitors, and fault responses.

Figure 12. Failure to spin up the reflector

Figure 13. Three GN&C fault monitors
StressTest is shown to avoid the failure mode Phasing Error. The test would be designed to show that no plausible stress of the GNC device management software could cause it to issue ill-timed commands or miss readings, thus arguing that the failure mode is not plausible.

With a complete set of test scenarios and analyses, the model can be used to find out:

1. which monitors, if any, do not have a test
2. which failure modes, if any, are neither avoided nor tolerated
3. which system fault responses are not tested
4. for a given system fault response, how is it tested (by what scenarios, analyses)

The answers give some measure of how well the verification goals are being met, as well as evaluating the completeness of mitigation of known failure modes (validation goal #3). All of these questions can be answered by generating matrices with appropriate parameters, and this is a built-in capability of MagicDraw (our UML/SysML tool).

For validation goals, the model can be used to find out which
failure modes are both unmitigated and do not appear in any fault tree, which could signify a missing scenario or branch in the fault tree analysis.

Completeness of the FMECA is more likely to be achieved if the FMECA is based on the complete an authoritative baseline of the system. Avoiding the FP team effectively creating its own system model by having a FMECA that is separate from the systems engineering baseline model improves the likelihood of achieving a comprehensive FMECA. This is one benefit of this model-based approach.

In our example, we’ve integrated the FMECA tightly with the general flight system model: as we’ve shown, the failure modes are directly associated with the components whose failures they describe. It would be possible to keep the general system model and the FMECA in separate models, which might be more convenient for a team in which the FP team was separate and distinct from the flight system systems engineering team. Model Relationships

Typically, the identification and modeling of the sets of monitors and local responses for each subsystem are assigned to different engineers. For this reason it is helpful to partition the monitor and response dictionary models by subsystem, as shown in Figure 18. In that diagram, each of the blue boxes represents a separate model, contained in a separate file, and owned by a different engineer. In many UML/SysML modeling tools, one model can “use” another model as a read-only library. This kind of usage is represented in the diagram as a dependency marked with the use stereotype.

So for example, the model “GNC Mon & Rsp Dct” would be owned by the systems engineer responsible for the GNC FP area, and it would contain all of the monitors and local response definitions for the GNC subsystem. All of the subsystem monitor and response models must use the SystemResponses model, because monitors must be mapped to system responses with the executes relation. Partitioning models like this can significantly enhance efficiency.

Matrices

It is important to know which failure modes are detected by fault monitors, and which failure modes are mitigated by responses. This information is in the relations between monitors and failure modes, between responses and failure modes, and between monitors and responses. These relationships are summarized in matrices which are generated automatically from the model. The detection matrix, showing a mapping from monitors to failures modes with the detects relationship, is shown in Figure 16. The rows are failure modes, the columns monitors, and each cell is marked with an arrow if the relationship is present. The matrix has totals of mappings, by package, for each monitor.

This matrix can be used to analyze how well failure modes are isolated. Each row can be thought of as a detection vector of 0’s and 1’s, with 1 corresponding to there being an arrow. If there is a unique vector value for a failure mode, then that mode can be clearly isolated. This corresponds to the isolated set concept of [1]. On the other hand, the pair of monitors Excess Attitude Estimation Error and Excess Rate Control Error both detect an overlapping set of failure modes, so this set of monitors is not sufficient to unambiguously identify the failure modes. It is not difficult to automate the analysis of the matrix to report on failure modes that are well isolated versus those that are not.

In the mitigation matrix shown in Figure 17, the rows contain failure modes, and the columns contain local and system-level responses as well as verification activities, because this variant of the mitigation matrix contains both types of mitigation: tolerates (at runtime), and avoids (design time). Arrows mark pairs where there’s a tolerates relation from the response to the failure mode, or where these’s a avoids relation to a test case or analysis.

It’s also easy to generate matrices with only one or the other kind of mitigation.

In our example, we’ve integrated the FMECA tightly with the general flight system model: as we’ll show, the failure modes are directly associated with the components whose failures they describe. It would be possible to keep the general system model and the FMECA in separate models, which might be more convenient for a team in which the FP team was separate and distinct from the flight system systems engineering team.
Figure 18. The network of fault protection models

The model entitled “Consolidated Monitor & Response Dictionary” pulls in all of the individual subsystem monitor and response models in order to generate a comprehensive system monitor and response dictionary. This product is needed as a source of definitions of FP commands and telemetry: there are commands to configure monitors (thresholds, etc), and to enable or disable monitors and responses. There are typically telemetry definitions for reporting monitor states, FP response actions, threshold values, and many more aspects of the FP implementation.

At the bottom of the tree is the FP Verification model, which contains the verification scenarios and the matrices. The matrices are derived from relationships distributed throughout the subsystem monitor and response models, the FMECA, fault trees, and system responses model. The verification model pulls all of this information together. (Note that the “use” relationship is transitive).

**Integration with Flight Software Engineering**

Flight software is responsible for implementing the lion’s share of fault protection. Fault monitors are almost always implemented by software. And while many local responses are built into the hardware, FSW typically implements many of them as well. System responses are normally entirely software based. Moreover, there is usually some sort of overall fault management executive function in the FSW that manages repair behaviors.

All of these tasks represent detailed requirements on FSW, and as for any other requirements on FSW, there must be auditable traces of the requirements to the design, and to tests. For FP, this means that all monitors must be mapped to their implementations and to tests, and all responses (local and system) must be mapped to design and tests.

As usual, we achieve these mappings using relations in the
model, tagged with stereotypes, and in this case the key one is implements. Figure 19 shows the use of this stereotype on dependencies from implementations of monitors and responses in the design to the definitions of these elements in the FP system models. For example, the SystemResponses model contains the definitions and specifications of the system fault responses, and SYS_MODE_1 response is shown residing in that model. The FSW FP Component Model is a software model of the specification and design of the FP component of the FSW, and therein reside the specification and design of the implementations of the system response behaviors, in particular a software component called SYS_MODE_1Impl, which is mapped to the SYS_MODE_1 response.

The figure shows another software model, the FSW GNC Model. It contains the specification and design of the FSW GNC component, and in particular the design of the implementation of the two fault monitors Device Overflow and Excess Rate Control Error (which we have seen previously in Figure 13). These fault monitor implementations are mapped to the monitor definitions in the GNC Mon & Rsp Dict model.

Part of FSW verification will involve generating matrices similar to those shown in Figures 15 & 16 that list the implements mappings to fault monitors in the FP subsystem models, and to system and local responses as well. These matrices make it easy to find monitors that have no implementation associated with them, for example. This kind of verification will help ensure that all monitors and responses have been implemented, and it will help in understanding the FSW design regarding where specific fault monitors and responses are implemented.

Figure 19 also shows the definitions of patterns and interfaces used in the implementation of fault monitors, system responses, and communications between monitors and the fault management function (that is the purpose of the FaultListener interface). These kind architectural features of the FSW will be influenced and informed by the system FP models, and this decreases the likelihood of errors in the implementation of FP in the FSW.

As we’ve mentioned, fault monitors have ground commands associated with them, commands such as setting thresholds of fault detection, setting frequency of checking, and enabling or disabling the ability of the monitor to cause a system fault response to be executed. All of these commands result in FSW interfaces. If these are expressed in the model, the code for their implementations can be generated partially automatically. I wouldn’t be difficult to make an auto-model generation capability that took the monitor definitions and generated the UML model for the design of the related command handlers. This could lead to significant improvements in accuracy and efficiency of producing the FSW.

Similar strategies could be employed for telemetry: fault monitors have telemetry associated with them: current value of the monitored variable, high and low water marks, no-
Figure 17. Fault monitor failure mode mitigation matrix

of thresholds being exceeded, reports of settings of thresholds, etc. The design of elements to collect and report these telemetry items could be generated from the fault protection models.

5. FUTURE WORK

We have only begun to scratch the surface of the potential of using modeling for the flight system fault protection. There are a great many possibilities in the representation, design and analysis of fault protection functionality using models.

The representational approach described in this paper is only one possible way to capture and reason about fault management functionality and off-nominal behavior. As we develop our ideas further, we intend to review and assess work that other researchers have been performing in this area. Development of representational patterns to describe off-nominal behavior is a non-trivial and difficult problem, and the expression of essential patterns will take the community of researchers to adequately resolve and apply.

One of the advantages of model-based techniques is the enhanced ability to analyze the problem at hand by analyzing the models of it. In addition to some of the simple model analyses we’ve shown (such as finding failure modes that are not detected by a fault monitor), there are a number of potential topics for analysis:

1. Finding failure modes that are not in a fault tree. If a failure mode appears in no fault tree scenario, it might be a reason to suspect that the set of fault scenarios, or the paths within them, are not complete.
2. Reporting propagation paths: where one effect causes another one or another failure mode. One of the difficult tasks of fault protection systems engineering is getting a complete picture of the propagation of faults throughout the system. The ability to search a model for chains of related effects could help make this task more tractable.
3. Validating propagation paths by mapping them to the physical model of the system, which should include electrical, mechanical, and data connections and pathways. This could serve both to ground the propagation paths in reality, and also to find missing propagation paths, based on physical paths that we not mapped to by any path identified in the analysis of the FMECA.
4. Likelihood of a failure mode could be determined/calculated from the likelihood of the set of associated causes. This information could be used to refine earlier estimates of failure mode likelihood as information about the design matures.

Expanding the modeling techniques and scope introduces still more possibilities. There is a systems modeling methodology called State Analysis, developed by R. Rasmussen and others (see [4]). This methodology involves building a State Effects model, a directed graph of the effects of states in the system on others. These models can be used to enable auto-generation of FMECAs and FTAs, and act as a cross-check on previously-generated FMECAs and FTAs (or alternatively, generate a failure propagation model by generating a set of directed graphs in the model in which a failure mode was the starting point, and the chain of produced effects would be a possibly branching, possibly even cyclic, network. This seems an area rich in possibilities for analysis of the model to validate the physical model, the FMECA, and the fault trees). This kind of automated generation not only reduces error, but also can be a check on the model itself. In addition, there are improvements to the representations described earlier that take into account some of the subtle aspects of system modeling and the relationships being captured. For example, we envision slightly more complex relations being used, that map monitors to effects, instead of failure modes, and also that map responses to causes instead of failure modes. We recognize that the current relations are simplified, but they represent current practice in the use of mitigation matrices. These new relations will be more precise and richer, but they will complicate the matrices described in this paper.

There are great potential benefits also in generating more software design artifacts from the fault protection models. For example, a fault monitor definition implies a set of related commands, as well as a set of related telemetry items. The implementation of the execution of these commands, and the production of these telemetry items, in FSW requires a considerable amount of software design and implementation, most of it following quite regular patterns. It would be very efficient, and reduce error considerably, if detailed design models, and indeed code, could be generated from the FP models. This is certainly doable.

Finally, we intend to apply some of the methodology we’ve describe here in more detail to an ongoing effort to describe and analyze behaviors and other aspects of an Earth-orbiter flight system currently under design at JPL. The exercise of generating a real model and the Fault Management artifacts
could provide valuable insights into how best to take advantage of this kind of approach.

6. SUMMARY
We hope this paper has given the reader a sense of the possibilities for these modeling techniques to improve the Fault Management systems engineering discipline. We have demonstrated how to realize the key fault management products - Failure Modes and Criticality Analyses, Fault Tree Analyses, Monitor & Response dictionaries, and FP architecture descriptions, using SysML and UML, and described some of the advantages to be gained by so doing. We believe these kinds of techniques have clear and significant potential for improving the precision, efficiency, and rigor of fault management.

ACKNOWLEDGMENTS
The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

BIOGRAPHY
Alex Murray
Alex Murray is a senior software engineer with the Jet Propulsion Laboratory, California Institute of Technology. He has led and done software development for flight, ground, and simulation software for missions and for technology development projects at JPL. Previously he led and developed software for a variety of projects at TRW (now Northrop-Grumman), and he served as a system engineer for the European weather satellite agency, Eumetsat, as well as software engineer for the Dresdner Bank in Frankfurt, Germany. His experience includes embedded and flight software development, prototype and research development, OS development, AI, analysis and simulation tools, science and image processing applications, business and GUI applications, and databases. He holds BS and MS degrees in mathematics from The Ohio State University.

John Day
John Day has over 20 years of systems engineering experience working on a variety of NASA projects. He has provided key technical expertise to a variety of human, deep-space and robotic observatory projects and programs (Constellation, Mars Science Lander, Mars Reconnaissance Orbiter, Deep Impact, Kepler, SIM, Spitzer, Cassini and Galileo). He has worked extensively in fault protection, and is actively involved in developing, extending and integrating the theory behind the practice of fault protection. The theoretical work that he has been developing has been captured in two recent papers. John’s experience includes working at his own systems engineering consulting company, at Lockheed Martin Missiles and Space, and at JPL.

Peter Meakin
Peter Meakin is the fault protection lead for the proposed Soil Moisture Active Passive (SMAP) mission, and has worked on a variety of projects at JPL. He has contributed to the fault protection design for SMAP, Cassini and Constellation as well as support for Mars Exploration Rover Pan Cam image calibration. In addition to his interest in fault protection he has worked in the area of attitude control systems (ACS) engineering, supporting the development of ACS architectures for over 50 mission concepts. He was the ACS engineer for JPLs Titan Saturn System Mission and Jupiter Europa Mission studies and supported SMAP and Cassini attitude control teams.