

# Modeling Off-Nominal Behavior in SysML

John C. Day<sup>1</sup>, Kenneth Donahue<sup>2</sup>, Michel Ingham<sup>3</sup>, Alex Kadesch<sup>4</sup>, Andrew K. Kennedy<sup>5</sup> and Ethan Post<sup>6</sup>  
*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109*

**Specification and development of fault management functionality in systems is performed in an *ad hoc* way - more of an art than a science. Improvements to system reliability, availability, safety and resilience will be limited without infusion of additional formality into the practice of fault management. Key to the formalization of fault management is a precise representation of off-nominal behavior. Using the upcoming Soil Moisture Active-Passive (SMAP) mission for source material, we have modeled the off-nominal behavior of the SMAP system during its initial spin-up activity, using the System Modeling Language (SysML). In the course of developing these models, we have developed generic patterns for capturing off-nominal behavior in SysML. We show how these patterns provide useful ways of reasoning about the system (e.g., checking for completeness and effectiveness) and allow the automatic generation of typical artifacts (e.g., success trees and FMECAs) used in system analyses.**

## I. Introduction

**W**E define off-nominal behavior as the unintended or unexpected behavior of a system. The process of considering the implications of off-nominal behavior in the development process and during system operation is referred-to as System Health Management (SHM)<sup>1</sup>. Fault Management (FM) is the subset of SHM that determines whether modifications or additions to system functionality, interfaces or components are necessary to prevent, mitigate or tolerate off-nominal performance of the system.

While often locally effective, the *ad hoc* methods used in FM result in gaps and inefficiencies in the overall SHM design. These methods are also unable to answer, or only partially address important characteristics such as the completeness and effectiveness of the SHM design. As a result, precise answers to system success in off-nominal situations are incompletely-known, with incomplete assessments of safety, reliability and availability based on time-consuming analyses and design processes that are based on multiple, often implicit, assumptions. As our designed systems grow in capability and complexity, the understanding of off-nominal behavior in these systems will become increasingly riddled with error and erroneous expectations, leading to systems that are actually less safe and reliable than systems fielded today. Only by increasing the rigor with which we consider system behavior as a whole – and off-nominal behavior in particular – can we improve our understanding of these systems, and make significant gains in safety, reliability and availability. Using the Systems Modeling Language (SysML) we show how some basic elements of FM can be performed rigorously, and particular artifacts derived from a system model.

## II. Modeling Off-Nominal Behavior

Systems are conceived, developed and designed with some defined purpose, expressed as a set of system objectives. The set of system objectives, over time, can be described as the intended behavior of the system. A realized system developed to accomplish these objectives has a resulting behavior that can be assessed to determine whether the system meets these objectives. This resulting behavior is determined from the design and current

---

<sup>1</sup> Technical Group Supervisor, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490, AIAA Senior Member.

<sup>2</sup> Software Systems Engineer, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490.

<sup>3</sup> Technical Group Supervisor, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490, AIAA Senior Member.

<sup>4</sup> Systems Engineer, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490.

<sup>5</sup> Systems Engineer, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490.

<sup>6</sup> Systems Engineer, Systems Engineering Section, 4800 Oak Grove Drive M/S 301-490.

operational state of the system. Off-nominal behavior is defined as the class of system behaviors that are outside of the boundaries of the intended behavior (based on purpose and objectives), or the expected behavior (based on performance of realized system).

Developing a system requires an understanding of both the nominal (intended and expected) behavior and the off-nominal behavior. However, all too often the focus is primarily on the nominal behavior, and the consideration of the off-nominal behavior is an afterthought. This is evident in many systems engineering process descriptions, that do not include (or only mention off-hand) guidance for consideration of off-nominal behavior. This leads to all kinds of poor design choices and compromised/over-complicated behavior<sup>2</sup>. Including off-nominal behavior in a system model provides a much richer and more complete understanding of the system behavior under all conditions. However, the off-nominal state space is much larger than the nominal state space – capturing this in a way that allows development of appropriate design mechanisms, without losing essential information, is difficult and time-consuming. What we define as off-nominal behavior is only a convenient characterization of a class of system behavior – it is only a defined subset of the possible set of state transitions within the system. When this subset is ignored or addressed incompletely, the design of the system, and the understanding of its behavior, is also necessarily incomplete.

Development of an integrated system model, used by the entire engineering team, has significant benefits over a document-centered approach to system development<sup>3</sup>. In this conception, typical engineering artifacts are views of the system model, instead of a set of documents with varying degrees of integration and connection to other documents. Typical FM artifacts, such as a failure modes and effects analysis (FMEA) or fault tree analysis (FTA) rely heavily on system descriptions such as block diagrams to define the components and sub-systems to which they refer. Development of an integrated system model that allows explicit connections between block diagrams and failure analysis artifacts has the potential to provide exceptional improvements in both the accuracy of the failure analyses and the amount of effort necessary to develop and maintain the analyses. Further, while it is possible and beneficial to merely include the results of failure analysis in the system model<sup>4</sup>, there is significant benefit to use the relations captured in the model to derive FM artifacts. The former allows explicit cross-checks between the failure analyses and the model structure, which significantly improves the ability of engineers to relate the two, but in the end is only an improved way of storing the information derived from the analysis. We believe that a much more powerful approach is to include the necessary relationships in the model, and to use these relationships to derive the necessary FM artifacts from the model. In this way, a FMEA or FTA becomes just another view of the system in the model, and is always current and consistent with the rest of the model. This approach enables significant improvements in both the accuracy and development time required. Other researchers have pursued a similar line of thinking by incorporating the necessary constructs in UML models<sup>5</sup>.

Modeling a system first requires an understanding of the specific purpose(s) the model is intended to support (i.e., what questions is the model intended to answer? ). By articulating the modeling need in this form, the content of the model and the effort to develop it can remain focused. Without a focus of this sort, the model can become bloated with unnecessary and irrelevant information that takes time and attention away from the salient information. In our work, we have focused on the following questions that are relevant for understanding and assessing off-nominal system behavior:

- 1) What is the intended purpose of the system? (Describe system objectives)
- 2) How are system objectives protected? (Describe intended alternate options/mechanisms to achieve objectives)
- 3) How are alternative options/mechanisms (behaviors?) implemented (Describe FM functions - detection of off-nominal behavior, identification of cause, determination of appropriate actions)
- 4) How are FM functions allocated to system components?
- 5) What are the effects of non-intended behavior in the system?

In the model we have developed, we have focused on development of concepts and relationships that allow us to reason about these questions and develop quantitative results. In this, we make use of mission ontologies developed at JPL by the Integrated Model-Centric Engineering (IMCE) team, that inform these relationships and basic SysML profiles that we utilize<sup>6</sup>. The general nature of SysML allows for many ways in which to represent concepts and relationships used to describe systems. We recognize that our solution is but one of many possible representations, but we have found significant utility in structuring our model in the manner described in the following section.

### III. Concepts and Relationships

The concepts and relationships that we have found to be essential to modeling off-nominal behavior are described in Fig. 1. In this figure, we show four basic concepts and the relationships of which we make use. These concepts are activity, component, goal, and state variable. We use activity to document intent, the actions that accomplish the intent of the system. In our model, we represent activity via SysML activity diagrams. Activity diagrams allow us to show the flow of intended action and the data transferred between the defined activities. Components are defined as blocks, and are intended to represent system elements at a given level of abstraction. Using the IMCE mission ontology, there is a specific set of relationships between function and component. Namely, that a component performs a function, and converse, that a given function is allocated to a component. In our relationship diagram, the activities we define can be thought of as an instance of a given function, so we apply the same relationship in our model. The set of system components are documented in a SysML block definition diagram, and swimlanes in the activity diagrams are used to show the allocation of activities to components. Our concept of activity contains a very important notion – goal definition – that allows us reason precisely about system intent and the boundary between intended and unintended behavior. In our conception, each activity is really a statement about the intended value of a given state variable. We do not yet explicitly include the notion of time in our model, but otherwise this definition is intended to match the conception of goal as defined in the state analysis methodology<sup>7</sup>. A goal, then, is a constraint on the value of a state variable over some time duration as expressed by the order of a set of activities. Each activity has a single goal associated with it, but there may be multiple goals associated with each state variable. We

relate each state variable to a component in the system, where each component typically has a set of state variables associated with it. This relation is captured in our model as a “characterizes” relations (as in “state variable X characterizes component A”). This relation is stated in this form, rather than the more direct “is a property of” relationship due to the multiple ways in which a set of state variables that describe a component could be derived. For example, depending on what specific information is necessary to capture, one could describe the temperature of a component as a single state variable (component A has temperature Y), or as a set of related state variables (component A has temperatures Y<sub>1</sub>, Y<sub>2</sub>, Y<sub>3</sub> and Y<sub>4</sub>). It is up to the modeler, and the questions being addressed by the model, to determine which of these characterizations are appropriate to include at a given level of model abstraction. These different characterizations are not incompatible, they are merely different ways of viewing the component (similar to the way different coordinate systems can be used to represent the same spatial realtion).

We also develop views of the model that allow us to relate these four concepts in different ways. The standard SysML views (activity diagrams, block definition diagrams, etc.) provide most of the utility we require, but we develop additional views to better describe and document other specific relations. In particular, we make use of a State-Effects Diagram (SED) to provide a graphical representation of the interrelations between state variables. The interrelations are depicted using the “affects” relationship, which is an abstraction of a mathematical constraint that couples state variables. The tail end of the relationship indicates the independent variable in the constraint, and the head indicates the dependent variable. As the relationship between variables may be complicated, a single mathematical constraint may represent multiple “affects” relationships. Figures 2 and 3 show a State-Effects Diagram and how one of its “affects” relationships is expressed as a mathematical constraint<sup>8</sup>.

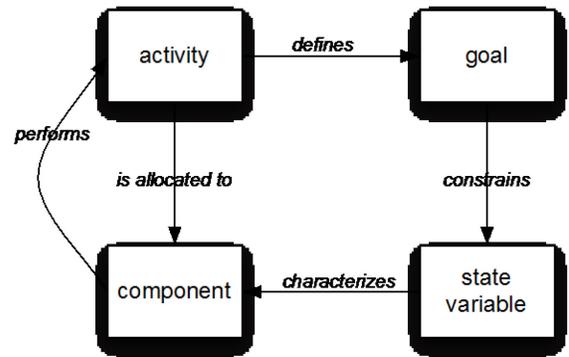


Figure 1. Concepts and Relationships

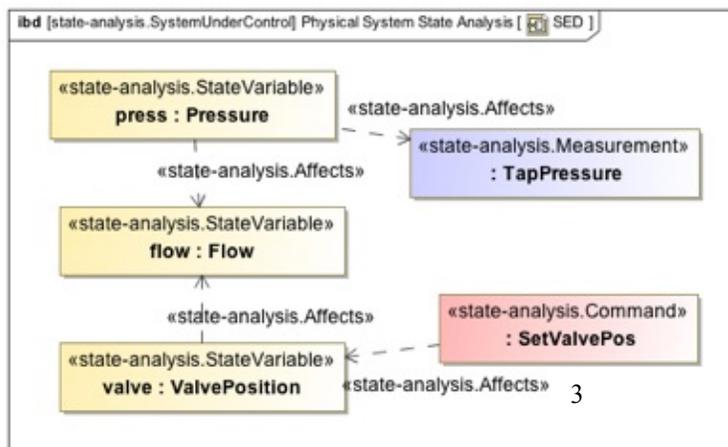


Figure 2. State Effects Represented in a SysML Internal Block Diagram

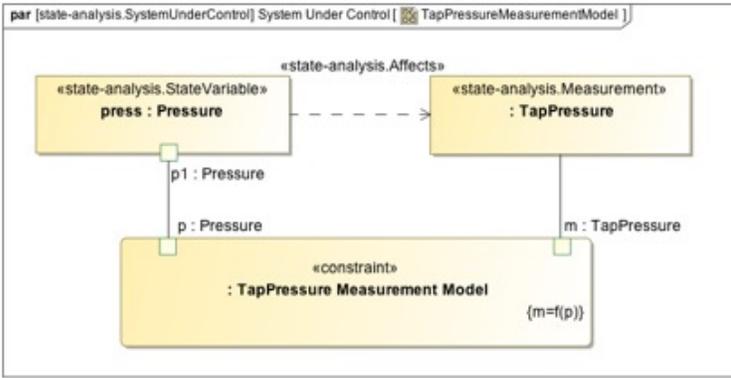


Figure 3. Affects Relationship Captured as a Constraint in a SysML Parametric Diagram

#### IV. Model of Behavior

In September of 2011, we began participating in a model-based systems engineering (MBSE) pilot to develop a limited-scope SysML model of the proposed Soil-Moisture Active-Passive (SMAP) mission. A significant part of this pilot was to develop a model of the off-nominal behavior of the SMAP mission. The focus of the effort was on the behavior of the mission during the spinup of a large radar antenna. A structural model of the components and subsystems of the SMAP Mission (flight system, instrument, telecommunications etc) was created, followed by the development of a model of the system behavior using activity diagrams. The structural and behavior models were integrated by

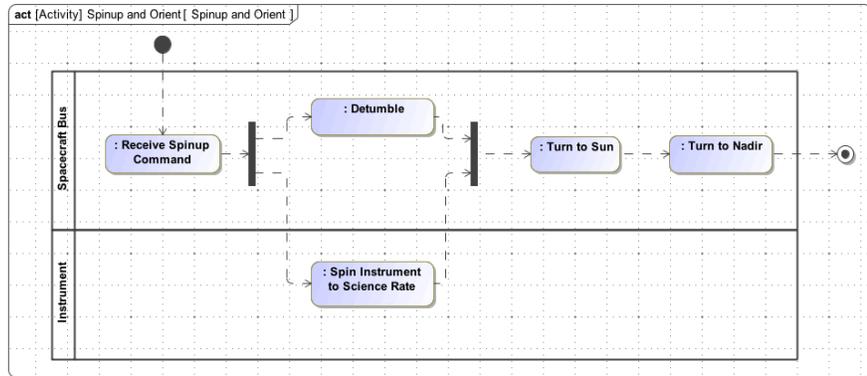


Figure 4. "Spinup and Orient" activity diagram

allocating functions to the components responsible for performing them (i.e. gyro performs attitude rate measurement). We modeled the nominal sequence of events leading to spin up as SysML activity diagrams, using swimlanes to allocate these activities to components. We were then able to decompose each activity into lower level activities and map these to lower level components responsible for performing them, allowing functional decomposition via an activity hierarchy.

An example activity to spinup an antenna is provided in Figure 4. The Spacecraft Bus performs four sub-activities, “Receive Spinup Command”, “Detumble”, “Turn to Sun”, and “Turn to Nadir”. The Instrument performs a single sub-activity, “Spin Instrument to Science Rate”. These sub-activities are allocated to the Spacecraft Bus and Instrument components, shown using swim-lanes. The relationships between the components of interest for this example are shown in the block diagram in Figure 5. Note that the Spacecraft Bus and the Instrument are both parts of the Flight System, but they are just a subset of the total set of parts, the others not being included for simplicity. The Reaction Wheels are components of the Spacecraft Bus, and the Instrument motor is part of the Instrument. As described earlier, each component is associated with a set of state variables of interest as well as a set of performed operations (e.g. activities that are allocated to the component via the swimlanes defined in the activity diagrams). In the model, each state variable is represented as a “reference property” of a component. Our choice to represent them as reference properties was a convenient way to show their relationship to the components.

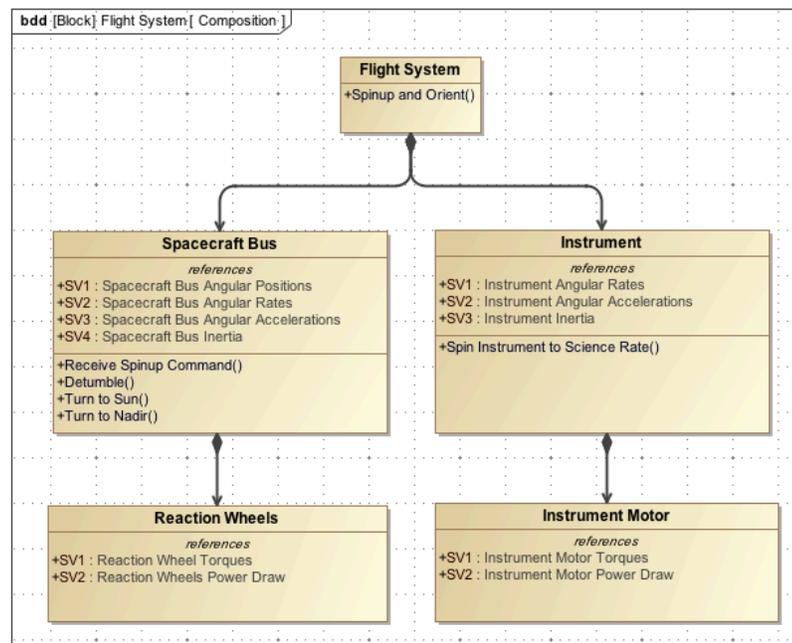


Figure 5. Block Definition Diagram for the Flight System

The state variables are a powerful means for understanding the causes and effects of failure modes. Figure 6 presents a notional state effects diagram for the components in the flight system. Using the relationships shown in this diagram, we can determine how one component’s state variables are affected by another’s. For example, we see that the angular rates (i.e. the three dimensional angular rate vector) of the Spacecraft Bus are affected by the Spacecraft Bus angular accelerations, which are in turn affected by the Spacecraft Bus inertia, the torques from the Reaction Wheels (“RW”s), and the Instrument’s angular accelerations. The torques from the RW’s are affected by the wheels’ power draw, and so on. The “affects” relationships shown between the Spacecraft Bus and Instrument angular accelerations are intended to be notional, representing the fact that the movement of one part of the spacecraft will affect the other. To be truly correct, additional interconnecting “affects” relationships would need to be drawn between the position, rate, and acceleration state variables for the spacecraft bus and the instrument. The

depth of the state effects diagram was limited here for simplicity, but it is evident that it can be elaborated much further. For example, we could connect the Power Draw state variables with other state variable from SMAP’s power subsystem, or we could connect the Spacecraft Bus inertia to any state variables of moving components elsewhere on the spacecraft. Thus we need not limit ourselves in scope to just looking at one subsystem on the spacecraft.

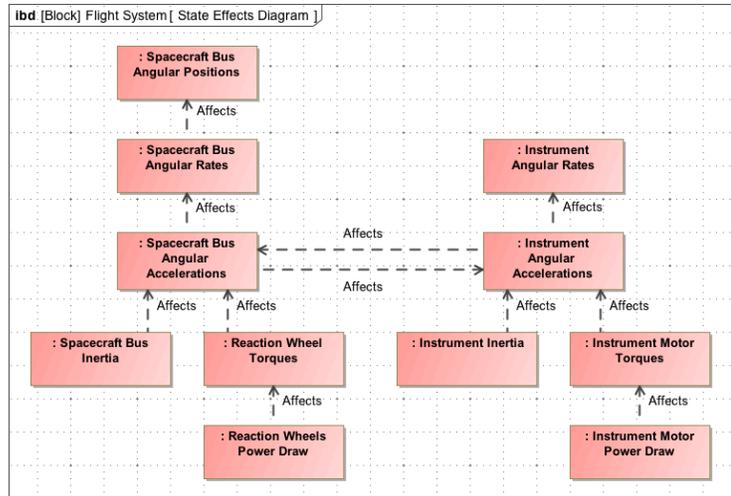


Figure 6: State Effects Diagram for the Flight System

## V. Application to Representative

### System

With the simple SMAP model defined above, we can begin to apply our defined concepts and relationships to derive information relevant to off-nominal system behavior. Characterization of the failure space is a key element of defining a fault management solution, and is necessary to determine coverage and completeness of a given fault management design solution. Typical methods for characterizing the failure space are Fault Tree analysis (FTA) and Failure Modes and Effects Analysis (FMEA). First we will show how a FMEA can be generated from the model.

In our model, every nominal activity is allocated to a component and each activity is associated with higher and/or lower level activities. To build the FMEA, we looked at each component and listed all the activities allocated to it. The identification of activities is shown, generically, in Figure 7. Each activity defines an intended outcome, and we define the inability to perform that activity as the defining characteristic of a failure mode. We applied this approach by using the logical negation of each activity (“Failure to Spin-up” instead of “Spin-up”) to define the set of failure modes for a each component of the system. These failure modes can be seen in Figure 8. This table organizes the failure modes as follows; each row lists the component of interest followed by a state variable then a failure mode. We include the state variable in our FMEA table to highlight the important role of state in our approach. Note that not all state variables associated with a given component are associated with a unique failure mode; if there are multiple activities that constrain a state variable, multiple failure modes will result. In this sense, a failure mode reflects an inability to constrain a state variable (or multiple state variables) in an intended way. A FMEA generated in this way is only as complete as the set of activities defined in the model. An incomplete model will result in an incomplete FMEA. The benefit of this approach, though, is that system functionality (intent) is directly linked to the FMEA, and changes to the system design or intended functionality change the FMEA in a precise and direct way.

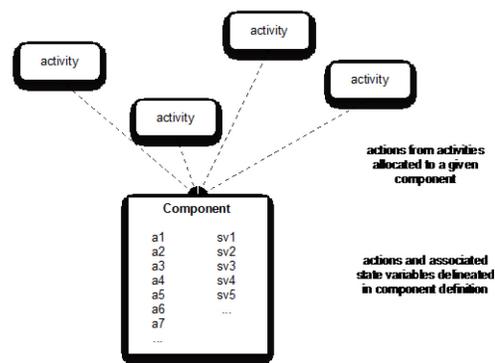


Figure 7. Collection of Activities Allocated to a Component

Identification of the failure modes is only the first step in the development of a FMEA. To make the FMEA complete we need to include cause and effect information. In general, the information contained within a FMEA is more accurate for immediate failure effect or failure symptoms, but often becomes 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<sup>9</sup>. We can use the SED to determine possible causes and effects of the failure modes in our FMEA. An FMEA is shown in Figure 9 with this information included.

Component	State Variable	Failure Mode
Spacecraft Bus	TBD	Failure to receive spinup command
Spacecraft Bus	Spacecraft Bus Angular Rate	Failure to detumble
Spacecraft Bus	Spacecraft Bus Angular Position	Failure to turn to sun
Spacecraft Bus	Spacecraft Bus Angular Position	Failure to turn to nadir
Instrument	Instrument Angular Rate	Failure to spin instrument to transition rate

Figure 8: Basic FMEA for SCB and instrument

Again, in this example we consider a failure mode to be an inability of a given activity to constrain a state variable to a desired range (over time?). We can determine causes for the state variable being out-of-range by observing what other state variables affect it. For example, the state variable associated with failure mode “Failure to Detumble” is “Spacecraft Bus Angular Rates”. These angular rates will only go out of range due to off-nominal accelerations. These accelerations could be caused by a lack of knowledge about the SCB inertia tensor, bad torques from the reaction wheels, or unexpected perturbing accelerations of the Instrument. We could continue to trace these effects threads, but we chose to only trace effects to either a leaf on the SED or first state variable after a change in abstraction level in the component hierarchy. Additional work is required to determine an approach that includes an appropriate level of causal information. Hence, causes for “Failure to Detumble” can be traced through its constrained state variable to the four state variables listed under “Causes” in the second row of Figure 9. If desired we can even trace the state variables all the way down to the Power Draw state variables. At this point however it suffices to use “RW’s unable to provide necessary torque” as a

Component	State Variable	Failure Mode	Causes	Effects
Spacecraft Bus	(undefined in model)	Failure to receive spinup command	(none for this example)	-RW unable to draw power -RW unable to provide sufficient torque -SCB unable to accelerate sufficiently
Spacecraft Bus	Spacecraft Bus Angular Rates	Failure to detumble	-SCB unable to provide necessary acceleration -SCB Inertia not sufficiently known -RW's unable to provide necessary torque -Instrument unable to accelerate nominally	-SCB unable to control angular position sufficiently
Spacecraft Bus	Spacecraft Bus Angular Positions	Failure to turn to sun	-SCB unable to provide necessary rates -SCB unable to provide necessary acceleration -SCB Inertia not sufficiently known -RW's unable to provide necessary torque -Instrument unable to accelerate nominally	(none for this example)
Spacecraft Bus	Spacecraft Bus Angular Positions	Failure to turn to nadir	-SCB unable to provide necessary rates -SCB unable to provide necessary acceleration -SCB Inertia not sufficiently known -RW's unable to provide necessary torque -Instrument unable to accelerate nominally	(none for this example)
Instrument	Instrument Angular Rates	Failure to spin instrument to transition rate	-Instrument unable to provide necessary acceleration -Instrument Inertia not sufficiently known -Instrument Motor unable to provide necessary torque -SCB unable to provide necessary acceleration	-Instrument unable to move nominally

Figure 9. Full FMEA Derived from Information in System Model

proximate cause. Note in this example we have only defined a failure mode for each defined activity. We expect that there is additional utility in including variants of failure modes, further distinguished by either the cause or the effect. For example, it is often useful in a FMEA to make a distinction of this sort to identify different responses that are intended to mitigate a given failure mode.

We can also use the same approach to determine the downstream effects of a failure mode by tracing state variable “affects” relationships. For “Failure to Detumble”, we see that the spacecraft bus angular positions are affected by our rates state variables, hence an effect of this failure would be that the spacecraft bus is unable to control its angular positions sufficiently well. This is only a notional example, considering that rates can be integrated to yield positions, but nonetheless it does illustrate that effects in the FMEA can be determined in exactly the same way as causes. Using the “affects” relation between state variables allows both an initial cut of component failure mode causes and effects; however in theory a completely exhaustive specification of the state effects within a overall system would allow the FMEA to be complete, capturing all possible causes and effects.

## VI. Future Research and Development

This work described here only scratches the surface of the possible applications of modeling off-nominal behavior. Determination of the relevant concepts and relationships needed to derive a FMEA only required a small set of relevant concepts and relationships. Other typical FM artifacts, and the ability to perform other analyses require additional concepts and relationships to be defined. In particular, we are interested in generation of fault trees, success trees and reliability block diagrams directly from the system model, and querying the information in the model to assess the completeness and effectiveness of the FM functionality. Ultimately, we intend to develop a SysML profile that captures these concepts and relationships and allows effective and consistent application within a system model.

We have performed some preliminary work in two areas – fault tree generation and representation of redundancy – that are important to our future goals. In each area, described in additional detail below, we have developed some initial insights and as well encountered some difficulties.

### A. Fault Tree Generation

In addition to the generation of an FMEA, one of our initial goals was the generation of a fault tree using the system behavior model. Our behavior model includes the nominal activities of the mission, so our first approach was to negate all the activities and turn this into our fault tree. We were able to write a script that could automatically do this by assuming that a failure of an activity leads to the failure of the higher-level activity. Each lower level activity in turn is composed of even lower level sub-activities that complete the leaves of the fault tree. Figure 10 shows the output of this script in XML. This approach gives a rough cut at a fault tree, but has a number of weaknesses. For one, we have not distinguished between activities that are necessary for mission success and simply desired. For this reason, every activity in our activity hierarchy shows up in the fault tree as an “or” leaf. A priority for us moving forward is to identify and distinguish these necessary activities from the non-necessary ones and apply this to our fault tree.

```
<element name="Spinup and Orient">
  <or>
    <element name="Turn to Nadir">
    </element>
    <element name="Detumble">
    </element>
    <element name="Receive Spinup Command">
    </element>
    <element name="Turn to Sun">
    </element>
    <element name="Spin Instrument to Science Rate">
    </element>
  </or>
</element>
```

Figure 10: XML Output of a Fault Tree—Failure of any activity leads to the failure at a higher level

This approach also leaves out a number of relationships we have in the model that can lead to a more complete fault tree. The three relationships we have identified, as shown in Figure 11, are the activity hierarchy, state elaboration, and component allocation. Our initial approach was to use the activity hierarchy, but as discussed above, this was insufficient. The state elaboration method takes advantage of the relationship different state variables have to one another. Each activity in the model is defined as a constraint on a state variable. We can use a state-effects diagram to show the relationship all of these states have to one another. Using these relationships, we are able to identify faults that lead to the failure of an activity. Each activity in the model is also allocated to a component, which is made up of sub-components. The failure of a component will cause an activity to fail, as will the failure of a sub-component. This allows a third way of showing the faults that make up a fault tree. Each approach by itself might fail to provide a complete fault tree, but each can be used as a cross-check on the others to find the complete set of faults. We have made some attempts to include the state elaborations and component allocations

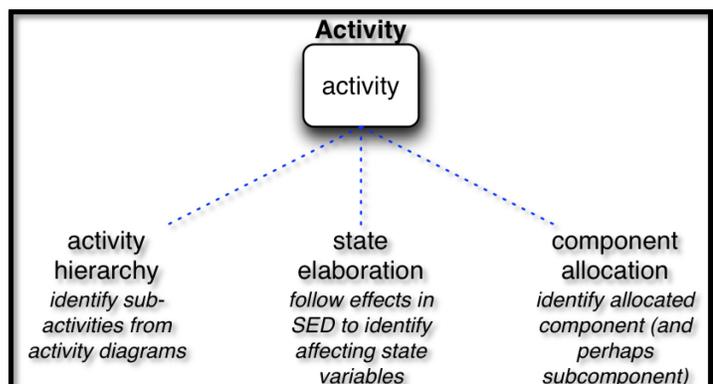


Figure 11: Types of faults that can cause the failure of an activity

in our fault tree, but have not yet formalized the effort. Our future work is to make a more complete fault tree using all three relationships to cross-check each other.

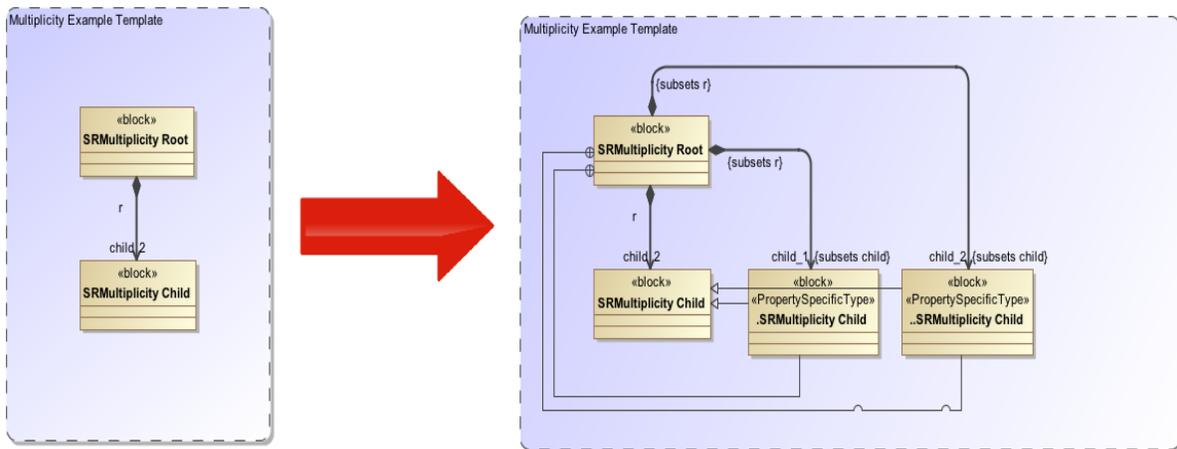
## B. Representation of Redundancy

Capturing system redundancies in a system model is crucial for FM as mission-critical hardware generally has a fully redundant assembly (two flight processors, extra reaction wheels, etc). Capturing physical redundancies is necessary for bookkeeping, like mass budgeting, but even more important for expressing contingency plans: if the flight processor experiences a failure, swap to the secondary flight processor. Although we are still working on formalisms for capturing functional redundancies, our pattern for capturing physical redundancies seems sound.

In SysML, physical redundancies are usually represented in the form of multiplicities. SysML has only a very limited expression of multiplicities. Multiplicities are a part of the SysML language that express, as a number pair, the minimum and maximum cardinality of a property. We can therefore create a SpacecraftBus (Bus) and a ReactionWheelAssembly (RWA) and say that Bus has exactly 4 RWAs.

Multiplicities are insufficient because if they are used, there is no way to indicate that RWA[2] is responsible for performing a behavior. Allocating a behavior to the type RWA allocates the behavior to all RWAs, and allocating instead to the part property RWA[4] allocates the behavior all 4 of our RWAs, neither of which is the desired allocation.

We decided that multiplicities were not a sufficient representation of redundancy in our model. We decided that we could not use multiplicities in our model. To solve this problem, we created a template RWA and created 4 specializations RWA1 through RWA4. This way, if we needed to allocate behavior to RWA1 specifically, we could. The RWA template had all properties and relationships that were common to all RWAs. Each of the RWA singletons specialized the parent and completely redefined all inherited properties (a process we are calling Symmetric Redefinition). The Multiplicity Pattern is the idea that the modeler should be able to represent a relationship and its cardinality and that the tool (via some script) should transform that multiplicity into  $n$  Part-Specific Types. Part-Specific Types are generated via a process called Symmetric Redefinition which recursively specializes and redefines all elements in a tree from a specific node (in this case, SRMultiplicityChild).



**Figure 1: The Multiplicity Pattern, Part-Specific Types and Symmetric Redefinition**

Unfortunately, the multiplicity solution is very modeling intensive. JPL has created a plugin to our modeling environment to aid in automating this process.

Our issues with multiplicities also made apparent the need for a Group. In SysML, there is currently no good way to show that a combination of elements is needed to do something. For example, any 3 of the 4 RWAs are needed to “Detumble” the Bus. Another missing concept we found was that of a Variant. Variants represent trade options and would be helpful for configuring analyses and documenting the rationale behind trades. For example, a FlightSystem with 5 RWAs. Groups and Variants are being discussed at OMG<sup>10</sup> and will (hopefully) be added to the SysML specification in a later release.

## VII. Conclusion

Fault Management is an essential part of the system engineering process that is limited in its effectiveness by the *ad hoc* nature of the applied approaches and methods. Providing a rigorous way to develop and describe off-nominal behavior is a necessary step in the improvement of fault management, and as a result, will enable safe, reliable and available systems even as system complexity increases. The basic concepts described in this paper provide a foundation to build a larger set of necessary concepts and relationships for precise modeling of off-nominal behavior, and a basis for incorporating these ideas into the overall systems engineering process. The simple FMEA example provided applies the modeling patterns we have developed and illustrates how the information in the model can be used to reason about the system and derive typical fault management artifacts. A key insight from the FMEA work was the utility of defining failure modes as the “inverse of intent”, and deriving this from the activity models. Additional work is planned to extend these ideas and capabilities to other types of relevant information and additional products.

## Acknowledgments

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We wish to acknowledge the the JPL Engineering Science Directorate, and Office of the Chief Engineer, for providing enabling support for the performing the MBSE pilot, SMAP flight system engineering team for their time and support, and the technical support of Sandy Friedenthal, Steven Jenkins and Robert Rasmussen in working out the essential concepts and appropriate modeling representations.

## References

---

<sup>1</sup> Johnson, Stephen B., and John C. Day, “Conceptual Framework for a Fault Management Design Methodology,” AIAA Infotech Conference, Atlanta, Georgia, April 2010; AIAA paper 227006.

<sup>2</sup> Rasmussen, R.D. 2008. “GN&C Fault Protection Fundamentals”, *31st Annual American Astronautical Society Guidance, Navigation, and Control Conference*, AAS 08-031, Breckenridge, Colorado, February 1-6, 2008.

<sup>3</sup> reference Bayer paper - Bayer, T.J., Bennett, M. ; Delp, C.L. ; Dvorak, D. ; Jenkins, J.S. ; Mandutianu, S, “Update - concept of operations for Integrated Model-Centric Engineering at JPL”, IEEE Aerospace Conference, Big Sky, Montana, March 2011.

<sup>4</sup> Day, John C., Murray, Alexander X., and Meakin, Peter, “Toward a Model-Based Approach to Flight System Fault Protection”, IEEE Aerospace Conference, Big Sky, MT, 2012

<sup>5</sup> Pai, G., Dugan, J. B., “Automatic Synthesis of Dynamic Fault Trees from UML System Models”, Proceedings of the 13th International Symposium on Software Reliability Engineering, 2002.

<sup>6</sup> Rouquette, N., Jenkins, S., “Transforming OWL2 Ontologies into Profiles Extending the SysML”, 12<sup>th</sup> NASA-EST Workshop on Product Data Exchange, Oslo, Norway, May 2010.

<sup>7</sup> Ingham, M., Rasmussen, R., Bennett, M., and Moncada, A., “Engineering Complex Embedded Systems with State Analysis and the Mission Data System”, AIAA Intelligent Systems Technical Conference. Chicago, IL. September 2004 .

<sup>8</sup> Bennett, Matthew B., Ingham, Michel, Jenkins, Steven, Karban, Robert, Rouquette, Nicolas, “Ontology for State Analysis: Formalizing the Mapping to SysML”, IEEE Aerospace Conference, Big Sky, MT, 2012

<sup>9</sup> Day, John C., Murray, Alexander X., and Meakin, Peter, “Toward a Model-Based Approach to Flight System Fault Protection”, IEEE Aerospace Conference, Big Sky, MT, 2012

<sup>10</sup> <http://www.omg.org/issues/sysml-rtf.open.html#Issue14827>