Failure Analysis and Products in a Model-Based Environment

Jean-Francois Castet, Magdy Bareh, Jeffery Nunes
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
castet@jpl.nasa.gov

Shira Okon, Larry Garner, Emmy Chacko, Michel Izygon
Tietronix Software Inc.
1331 Gemini St # 300
Houston, TX 77058
mizygon@tietronix.com

***Abstract*—The work presented in this paper describes an approach, including a methodology and tools, which allows system engineers to capture failure-related information in a model and generate automatically key failure analysis products: the Failure Modes, Effects and Criticality Analysis (FMECA) and the Fault Tree Analysis (FTA). The work has been developed by Tietronix Software, Inc. and the NASA's Jet Propulsion Laboratory (JPL), and the resulting auto-generated artifacts shown in this paper demonstrate the ability to obtain powerful reliability and fault management products in a model-based environment.**

TABLE OF CONTENTS

1. INTRODUCTION

As part of the process to create failure analysis products, the reliability engineer needs to gather existing project data, such as a list of components, their associated functions, and various diagrams showing their interaction. These elements are usually defined and documented by other engineers on the project, and gathering all of the required data can become inefficient and time-consuming. As a consequence, failure analysis artifacts can rapidly diverge from the design as it evolves, and become obsolete. In a model-based environment, where this data is available in a common repository, the automation of data gathering and processing results in freeing time so that the engineer can focus on performing the analysis. This approach ensures completeness of the analysis and consistency with the current design, a task particularly difficult using traditional methods in the context of growing system complexity.

This innovative approach is based on the recently-published JPL's Fault Management ontology [1], which contains the necessary concepts to capture failure modes, faults and failure propagation. This ontology extends the Behavior ontology [2], and is part of a series of ontologies developed at JPL under the Integrated Model-Centric Engineering (IMCE) initiative to guide standard modeling across projects. The common vocabulary enables engineers to capture problematic behaviors to support the discovery of failure paths in the system, discovery that often relies on engineer-to-engineer communication and data spread across multiple documents. An additional benefit of this approach is to capture failure and failure propagation information in the same central repository, with "nominal" behaviors of the system (rather than in separate artifacts managed by separate teams), resulting in the earlier discovery of design or operational issues.

We present in this paper a methodology and a software toolset which makes use of the Fault Management ontology. The objective of this effort was to obtain auto-generated artifacts that match the expectations of products developed through traditional methods and standards, so that engineers could work with familiar products. When deemed useful, additional pieces of information or representations enabled by the analysis of the model were included, to aid engineers in analyzing failure modes and effects. The software extracts the relevant behavioral information for each system artifact, from a common model, and converts this information into the appropriate format. In that light, JPL and Tietronix Software designed an approach to map the needed information to generate a FMECA and a Fault Tree from a SysML model consistent with the Fault Management ontology, as shown in Section 2. Section 3 further describes the underlying approach, and the technical implementation by Tietronix Software of the MagicDraw plugins for FMECA and Fault Trees generation, their features and user interface, their model validation capabilities, as well as the different outputs generated. The outputs range from spreadsheets to graphical representations, as well as file formats intended as basis for interchange with other third-party tools used in the reliability and fault management fields. These plugins were used on several models with different complexity across different projects, with some illustrative examples presented in Section 4. Section 5 concludes this paper.

978-1-5386-2014-4/18/$31.00 ©2018 IEEE

2. Ontology and Artifact Generation Approach

Failure Artifact Description

This paper focuses on two failure analyses: the Failure Modes, Effects and Criticality Analysis (FMECA) and the Fault Tree Analysis (FTA).

The FMECA is a reliability design analysis technique used to analyze systematically postulated failures in components of the system to determine the resultant effects. It is performed primarily to identify potential design deficiencies and single-point failures (SPFs), so that appropriate risk-management steps may be undertaken for the flight design and mission operations (e.g., preventive measures to decrease the likelihood of occurrence of the failure, mitigation actions to lessen the consequences of a failure when present in the system). Several variants of the FMECAs exist, e.g., functional and piece-part FMECAs. Functional FMECAs are performed down to the functional level to evaluate the impact that lower-level failures have on system operation. Functional FMECAs can be done at different levels of the design hierarchy, where the scope of the FMECAs can be the entire system, or specific subsystems or assemblies. Piece-part FMECAs are performed at interfaces to ensure that irreversible physical and/or functional damage does not propagate across Fault Containment Region boundaries. For support equipment, FMEAs are performed at the piece-part level on interfaces to verify that support equipment failures cannot propagate or cause damage to flight hardware. This paper focuses on functional FMECAs, but the methodology presented in this paper can be extended to generate other variants. When performing the FMECA, all components are operating within specification except for the failure mode under consideration. Each failure mode postulated is considered to be the only failure in the system (i.e., it is a single-failure analysis).

The FTA is a top-down reliability design analysis technique to deduce faults (i.e., failure causes) systematically and to assess coverage of risk-reduction steps for those failure causes. It covers a broad scope of threats beyond hardware failures, such as environmental effects or operator errors. Contrary to the FMECA, it assesses the impact of combinations of threats, including common-cause failures. The top-down approach of the FTA is complementary to the bottom-up approach of the FMECA.

Many of the failure analysis products rely on the same information (e.g., failure mode, failure propagation). As a consequence, one of the guiding principles for establishing the methodology to auto-generate these artifacts was to avoid modeling the artifacts themselves, but rather to model the system, including its off-nominal behavioral aspects, and then to query and extract the appropriate information to create the artifacts of interest. In essence, model once, then filter and present the relevant information. To this effect, JPL recently published a Fault Management ontology, which defines a common vocabulary to capture problematic behaviors, in particular failure modes, faults, and failure propagation. Behaviors are typically modeled using state machine representations (e.g., states, transitions), or mathematical expressions (e.g., constraint blocks). The section below briefly summarizes the Fault Management ontology. For details about the ontology or the diagram conventions, the reader is referred to [1].

Ontology and Mapping to the Artifacts

Figure 1 shows in blue color parts of the Fault Management ontology relevant to this paper, with some updates since its initial publication in [1]. Failure modes are considered as undesirable behaviors depending on the functional context (context provided by the PerformanceConstraint concept), and ViolationExplanations are the conduit to indicate their presence in the system. Fault or failure propagation is modeled using CauseExplanations, a construct that indicates logical causation between behaviors. One modification from the initial ontology was to redirect the relationship between PerformanceConstraint and Function to the "performs" relationship instead, to allow for a better modeling of redundancy in the model. Also, the concepts of Likelihood, MissionImpact, DetectionMechanism and Mitigation have been added to the ontology. Part of the work done to auto-generate failure artifacts was to map the concepts and relationships present in the ontology to the information needed to construct the artifacts. This mapping is presented next.

Failure Modes, Effects and Criticality Analysis (FMECA)

The FMECA table is comprised of several columns. The exact column number and label vary depending of the standard used (e.g., IEC-60812 [3], former MIL-STD-1629A [4]), but all standards address the following concepts: component affected by the failure mode of interest; the failure mode; its cause, and its effects at different level of abstraction (e.g., local effect, end effect) in a given mode of operation; ways of detecting and mitigating the failure mode. The criticality analysis related to the failure indicates its likelihood of occurring and the consequence on the mission. A typical table is shown in Figure 2.

Failure information is captured in a model according to the Fault Management ontology, and the following queries are performed to create a FMECA table that matches expectations of traditional standards mentioned above:

- *Failure mode*: a ViolationExplanation in the model indicates the presence of a failure mode in a certain context (functionality). Each ViolationExplanation in the model will have a dedicated row in the FMECA and the name of the ViolationExplanation will be reported in the "failure mode" column.

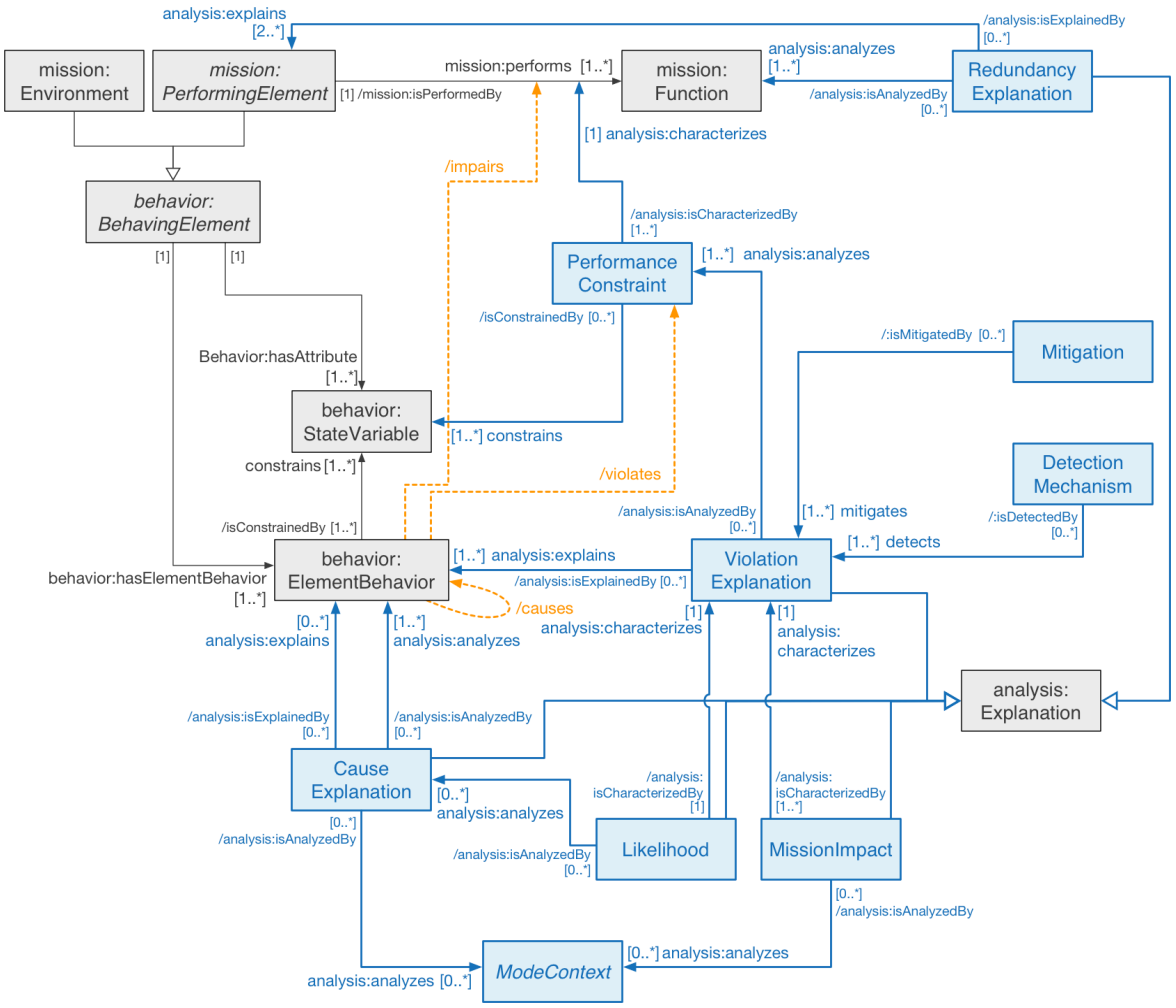


Figure 1. Excerpts from the Fault Management Ontology [1] – *Concepts shown in boxes, with abstract concepts italicized; directed arrows for relationships between concepts (name next to the target concept); derived relationships are preceded by "/"; multiplicity indicated between brackets.*

- *Component*: as shown in the ontology, a ViolationExplanation explains one or more behaviors associated with a unique component. Another method can be used to find the associated component: a ViolationExplanation analyzes at least one PerformanceConstraint that characterizes a "performs" relationship between a Component and a Function. The Component is then listed in the "component" column of the FMECA. Note that the component hierarchy can be indicated in the FMECA using the relationship between components (e.g., composite association, reference association).

- *Function*: as mentioned above, a ViolationExplanation relates to at least one Function through the PerformanceConstraint concept. The identified Function is then listed in the "function" column.

- *Cause*: Causes are captured in the model using CauseExplanations. The name of all CauseExplanations that analyze the same behavior as the ViolationExplanation of interest explains are reported in the "cause" column.

- *Phase of Criticality or Operational Mode of Equipment*: as part of the model, it is possible to define some context in which some functionality is required. The current implementation of the plugin is restricted to high-level "mission phases" (for example: launch, cruise, orbit insertion, science collection, disposal. Other phases can be defined based on the system and the mission). Functions are related to Mission Phases through a composite association. If there is no relationship, it is assumed that the Function is needed in all phases. The Mission Phases found through the query are reported in the "phase" column. It is envisioned in the future to extend this concept to other contextual information

by leveraging the ModeContext concept, such as operational modes of the system, or to potentially infer the context through other means rather than direct relationships in the model.

- *Local Effects*: failure propagation is captured in the model using CauseExplanations. As indicated in [1], CauseExplanations indicate a causal relationship between behaviors: for example, if a CauseExplanation analyzes Behavior 2 (B2) and explains Behavior 1 (B1), it means that B1 is the *cause* of B2, or conversely, B2 is the *effect* of B1. The latter interpretation is the one of interest for determining the effects. At JPL, we decided that the local effect would represent the effect of a failure mode on the behavior of components one level up in the component hierarchy. Using the example described in [1] where a Lamp is part of an IlluminationDevice that is itself part of Rover, if a failure mode in the Lamp has an effect on the IlluminationDevice, then the associated behavior in the IlluminationDevice is reported in the "local effect" column. If the behavior of the IlluminationDevice is itself explained by a ViolationExplanation, the ViolationExplanation is reported in the "local effect" column instead of the behavior. Note that it is possible to have several effects listed in the "local effect" column, depending on the causal relationships captured in the model through CauseExplanations. Note also that it is possible for a failure mode to have no local effect if no higher-level components are reached through causal relationships. As mentioned earlier, the FMECA is a single failure analysis: this means that the effects reported in the "local effects" column must be related to a single ViolationExplanation. If several ViolationExplanations need to be combined to obtain that effect (e.g., through an AND logical statement in a CauseExplanation in the fault propagation path), then that effect will not be listed.

- *End Effects*: continuing the discussion from the local effect above, the end effect of a failure mode is determined through causal relationships using CauseExplanations. Given that FMECAs can be performed at different levels (e.g., system, subsystem), it was decided to leave the determination of the level to the user (in our implementation, through a selection menu in the FMECA plugin): the user selects the top component, and end effects will be reported in a similar fashion as for the local effects: if there is a chain of CauseExplanations that reaches the top component, then the behavior of the top component or its ViolationExplanation if applicable will be reported in the "end effects" column. For example, the user can select the Rover as the scope of the FMECA: in that case, given that a broken lamp results in the inability for the Rover to perform some science activities, that effect will be listed in the row related to the broken lamp. Note that in this example, the fact that there is a single level "hop" between the local and the end effect is coincidental; many levels can exist in general. Also note that the comment made above for local effects regarding the single failure point of view of FMECA also applies for determining end effects.

- *Detection Method*: in the ontology, there is a relationship between a ViolationExplanation and DetectionMechanism. Found Detection Mechanisms will be listed in the "detection method" column.

- *Compensating Provisions*: using the ontology, mitigating actions can be captured using the Mitigation concept, related to ViolationExplanation through the "mitigates" relationships. Found Mitigations will be listed in the "compensating provisions" column.

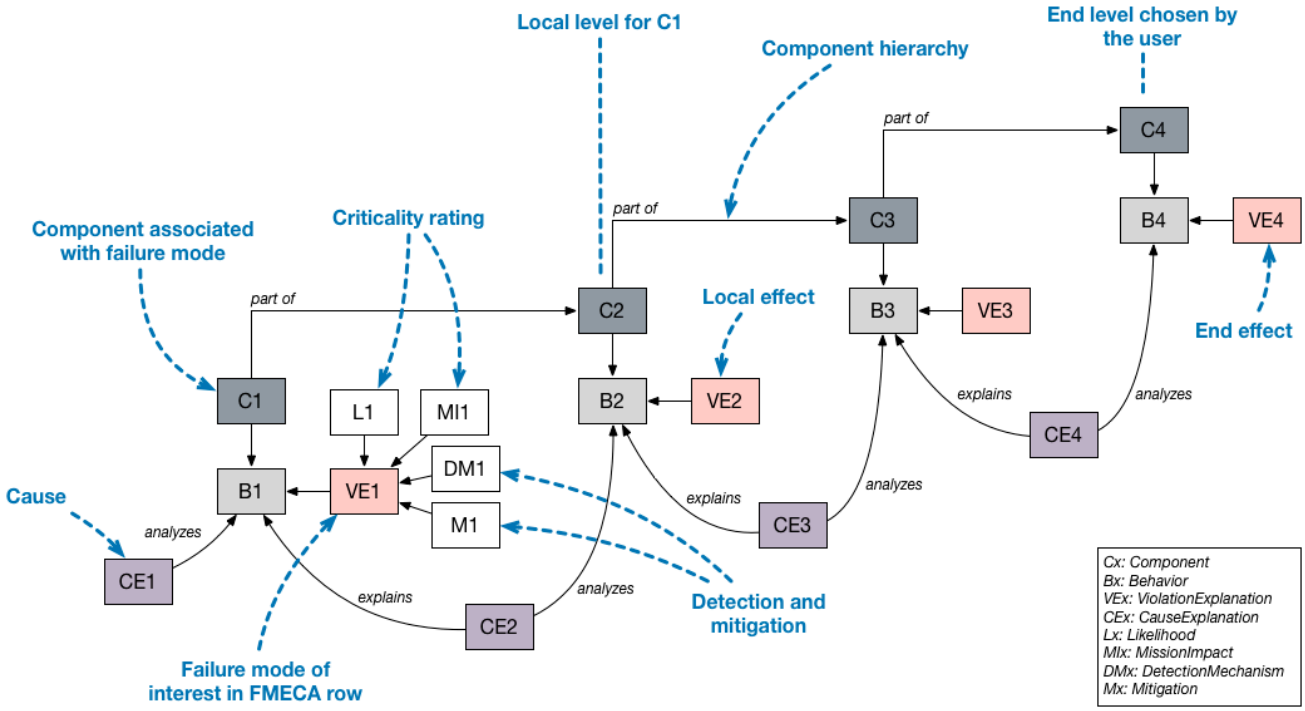
- *Likelihood*: the likelihood of a failure mode is captured in the model using a Likelihood block that characterizes the ViolationExplanation of interest. Different standards and approaches are possible for rating the likelihood. At JPL, we use a 1-3 rating to indicate a low, medium, or high probability.

- *Severity/Mission Impact*: as for the likelihood, the severity of the failure mode is modeled using a MissionImpact block that characterizes the ViolationExplanation. Here as well several rating schemes are used in the industry. JPL uses a 1-6 rating scheme, where 1 is a minor impact on the mission and 6 is a catastrophic failure. If no rating is provided, the tool will try to infer a rating, by making use of the redundancy information captured using the RedundancyExplanation concept.

Figure 2 illustrates the building of an FMECA table, in a simple case. One more piece of information the tool presents is the entire fault propagation path starting with the ViolationExplanation of interest. This is a great enhancement provided by the model compared to the traditional artifact, as it allows the analyst to fully understand the impact of a failure mode in the system, rather than at two or three pre-determined levels.

Fault Tree Analysis (FTA)

The FMECA process described above is fairly straightforward compared to the Fault Tree construction. Despite the existence of detailed standards or handbooks related to building fault trees (e.g., [5]), the determination of the immediate causes of an event and their grouping often remain dependent on the analysts and their understanding of the system. As part of the development of the FTA plugin, we decided on an approach to build systematically a fault tree from model information and developed rules to present and to group events.



Item			Failure		Effects Pre-Mitigation (e.g. no redundancy)			Mitigation		Criticality Post Mitigation	
System / Tier 1 / Tier 2 / Tier 3		Function Description	Mode	Most Probable Cause	Phase of Criticality or Operational Mode of Equipment	Local Effects (Assembly/Subsystem)	End Effects (System/Mission)	Failure Mode Detection	Compensating Provisions	Likelihood [1-3]	Mission Impact [1-6]
C4											
C3											
C2											
	C1	C1 function(s), not shown in model above	VE1	CE1	All, not shown in model above	VE2	VE4	DM1	M1	L1	MI1

Figure 2. FMECA Construction from Model Information

The Fault Tree Analysis starts with determining the scope of the analysis. To that effect, the analyst is presented with a list of all ViolationExplanations in the system (organized by components). The analyst then selects the ViolationExplanation that will constitute the top of the tree. From there, the tool will explore systematically the fault propagation paths in the model by identifying CauseExplanations and parsing the logical statements they may hold. The immediate causes of an event are determined through various means, as described in the next few paragraphs.

The simplest case occurs when "simple" CauseExplanations are present, that is, CauseExplanations with single analyzes and explains relationships. For example, as illustrated in Figure 3, three CauseExplanations analyze the same behavior (B1), and each explains a single behavior (B2, B3, and B4). In that particular case, the fault tree will interpret this model as an OR gate with three immediate causes of the event of interest. If the behaviors are explained by ViolationExplanations, the ViolationExplanations are reported in the tree instead of the behaviors themselves. Note that the same tree could be captured using a single CauseExplanation with the following logical statement: B2 OR B3 OR B4.

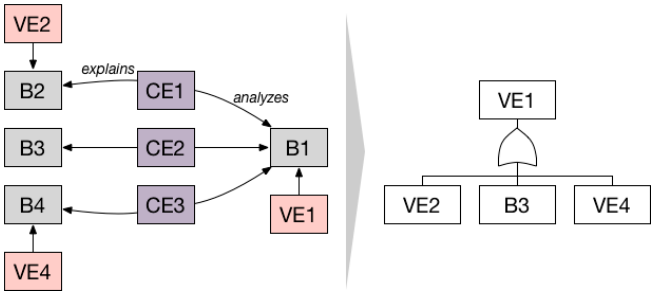


Figure 3. Simple OR gate

However, the logical combination of causes can be more complex than OR statements. Logical statements are captured using UML expressions that reference a library of logical operators and the relevant behaviors in the model. For example, if Behavior 2 (B2) and either Behavior 3 (B3) or 4 (B4) are needed for a given effect (B1), then the logical statement captured in the model will be of the form "B2 AND (B3 OR B4)". The tool described in this paper is able

to understand these logical expressions, and renders them in the form of a tree. However, due to some restrictions on the structure of fault trees for this approach to comply with the NASA Fault Tree Handbook [5], more processing is required. Indeed, fault trees must alternate gates and events; gate-to-gate connections or event-to-event connections are not allowed. In the example above, the tool cannot directly connect the AND gate and the OR gate; an event must be placed as shown in Figure 4, and the tool has been designed to extract labels from the model for these "extra" events as much as possible. It currently uses for example the names of expressions in the expression tree modeling the logical statements. More techniques are considered for future enhancements, such as leveraging super-state information.

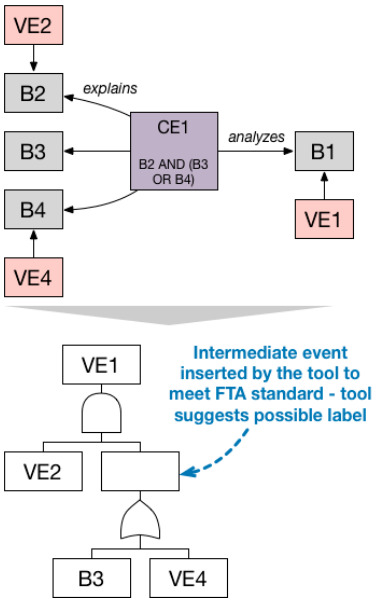


Figure 4. Logical Expression

The tool also handles common-cause failures. As described in [1], common-cause failures are modeled using CauseExplanations with multiple analyzes relationships. In that case, the tool will display the associated cause in multiple locations in the tree, with the "CC" label following the name of the event, as shown in Figure 5.

Fault tree standards provide guidance for determining the causes of an event, particularly highlighting the necessity to take a methodical one-step-at-a-time leading to the discovery of immediate causes, without jumping to lower level events. For example, if several failure modes of a same component lead to the same effect, they will report first a "component failure" in the tree before going into the detail of the various failure modes. The tool provides a similar functionality by identifying situations where multiple failure modes of a same component are found as immediate causes, and inserts an event in-between highlighting the component. A similar behavior occurs when several failure modes related to a same function are found: in that case, an event related to the function is inserted in the tree, as shown in Figure 6.

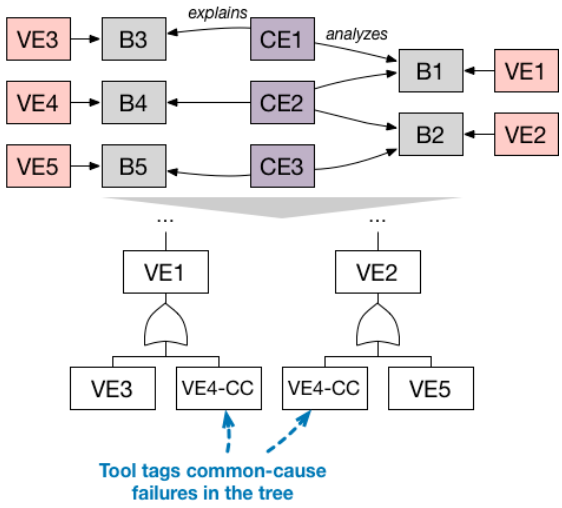


Figure 5. Common-cause failures

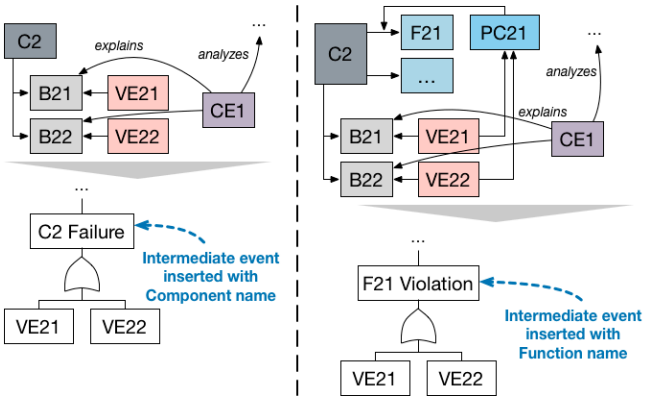


Figure 6. Component and Functional Grouping

The FTA plugin traverses the failure propagation path usually by looking exclusively at CauseExplanation relationships. However, more model constructs are considered to deduce the propagation path. For example, the tool takes into consideration transitions between states in state machines. While building the fault tree, if there is an incoming transition to an event in the tree, and the transition itself or the source of the transition have CauseExplanations pointing at it, then the tool will continue exploring the failure path using these CauseExplanations. More constructs are under consideration for future releases, such as continuing the failure path from sub-states to super-states.

Finally, the tool identifies parts of the tree that are repeated in different branches and creates transfer gates and sub-trees for ease of review.

JPL and Tietronix Software derived about 90 high-level requirements for the FMECA and FTA tools, specifying both the tool scope, but also the required outputs and user interface. The following section describes the software implementation.

3. SOFTWARE IMPLEMENTATION

Tietronix Software, Inc. developed a set of Fault Management (FM) tools within the selected modeling tool (MagicDraw) to extract the information captured within the models. Tietronix Software developed a FMECA plugin, a FTA plugin, and error handling capabilities for both the FMECA and FTA plugins. Tietronix Software extended a methodology established in prior work for nominal model-based design to support the fault management domain [6], [7], and [8]. Tietronix Software developed a set of FM plugins that were demonstrated on NASA JSC projects for the Cascade Distillation System and a Habitat Power Architecture model [9]. Tietronix Software leveraged this technology and adapted the tools to fit the Fault Management ontology and rules presented in Section 2 to automatically extract FMECA and Fault Tree Analysis products from the system models. The error handling capability checks the models against the Fault Management ontology and flags non-conformances. Both plugins include this capability and generate a report with the suspected model element or relationship and a description of the suspected non-conformance. If errors are detected, the analyst is given the choice to continue with the generation of the artifact, or to stop and fix the model. The error detection capability gives the assurance to the analyst that the model is built correctly according to the rules of the Fault Management ontology.

As part of the plugin development a Fault Common plugin was developed. This plugin captures the mapping between the ontology (Figure 1) to the MagicDraw SysML elements and relationships. The Fault Common plugin is a utility that is used by both the FTA and FMECA plugins, making it possible to make updates reflected in the ontology in one place. The software is written in JAVA and its installation adheres to the MagicDraw plugin installation standards. This section describes in detail the tools developed.

FMECA Plugin

The FMECA Plugin was built to extract details about potential failure modes described in the model and traverses behavior constructs to determine potential local/end effects for analysis.

The FMECA plugin presents a user interface that allows the user to select the end effect level based on the component hierarchy. Using the ontology described in Section 2, the plugin identifies all the ViolationExplanations in the model, all Components and associated hierarchy (both physical through composite associations and logical through reference associations), and the behavior elements. The tool then traverses the fault propagation paths, extracting data, to populate the FMECA table (Figure 2).

Several outputs are generated, and presented to the user through different formats:

- *Output views*: two views are generated by the tool, a detailed view and a summary view. The detailed view has a unique row for every combination of component, failure mode, cause, mission phase, local effect, end effect, detection method, and mitigation detected along each fault propagation path. The detailed view is an additional tool provided to the user to understand the model data for a more in-depth failure analysis and for debugging the model. By contrast, the summary view has a row for every combination of component, failure mode, and mission phase, but compresses the data for causes, local and end effects, detection methods, and mitigations into a cell for each row, akin to a traditional FMECA output.

- *Output formats*: The results are displayed within the plugin's user interface in tabular form, allowing the user to look at the FMECA in the modeling environment and verify the model correctness. The user interface is designed to allow the customer to tailor the desired product by selecting or deselecting columns of interest for analysis. The analysis is saved in Excel and csv file formats, for communication, review and archiving. The plugin is able to support any pre-defined FMECA Excel template.

FTA Plugin

The FTA Plugin derives fault trees from ViolationExplanations, representing potential top-level events, and traverses relationships to extract the fault tree event paths for analysis. Using the ontology described in Section 2, the plugin identifies all the ViolationExplanations in the model and lists them as potential events for analysis in the Fault Tree User Interface. The plugin then generates a Fault Tree Markup Language (FTML) file. The FTML is an XML-based file format designed by Tietronix Software to generate the fault tree displayed in the Fault Tree Viewer in MagicDraw, an Excel version of the fault tree, an SVG view, JPG file outputs, as well as to support third-party tool exchange. For example, using an adapter reading FTML data, the fault tree was exported to SAPHIRE, a probabilistic risk and reliability assessment tool [10].

The user interface in the Fault Tree Viewer allows the modeler to interact between the fault tree display and the MagicDraw model elements by right-clicking on a Fault Tree event and selecting "Select in Containment Tree" (see Figure 7). The user can quickly explore the fault tree and understand the elements that participated in its construction. In addition, the user can collapse and expand events in the tree and allow for traversing multiple linked fault trees via transfer events.

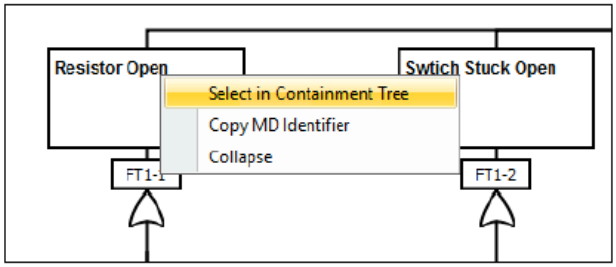


Figure 7. Fault Tree Event Tool Options

As mentioned above, the plugin also generates a version of the fault tree in a SVG format, that can be opened in a web browser for example, enabling the communication of the fault tree to colleagues and reviewers in a cross-platform fashion. The SVG view is also fully interactive, with collapsing/expanding actions and transfer event navigation. Excel versions of the fault tree can be used for capturing information in subsequent fault management activities, and the JPG files can be used in reports or presentations.

Transfer events are automatically generated when repetitive events are found in the tree. The tool provides the ability to navigate between fault trees via transfer events (shaded in blue and marked with a standard triangular symbol) and back via selection of a fault tree in the "Transfer From" table. In Figure 8, the left column provides the user a list of available views generated for one user selected event. The options include the complete fault tree which does not show transfer events or the fault trees with active transfer events.

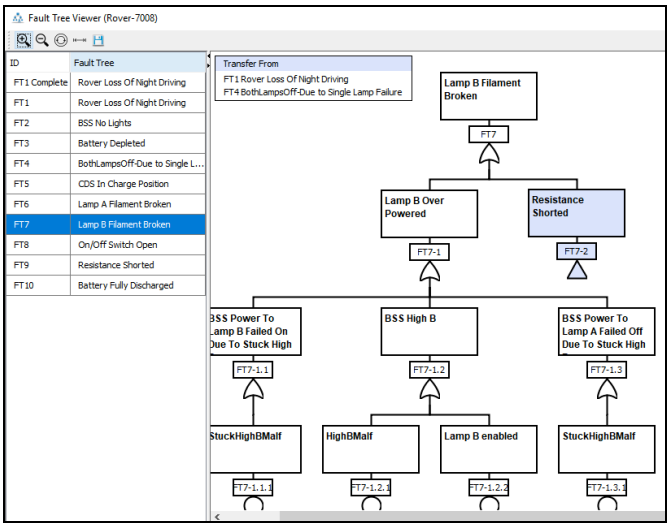


Figure 8. "Lamp B Filament Broken" Transfer Tree and "Resistance Shorted" Transfer Event

The tool is capable of extracting statistics from the model including counts for Basic Events (unique/ repeated/ total counts), Intermediate Events (unique/ repeated/ total counts), Transfer Events (unique/ repeated / total counts), Common Cause failures, and occurrences of Basic Events.

4. EXAMPLES

The following section describes example models used to demonstrate the modeling techniques and toolsets.

Illumination Device Example

The Illumination Device Model example represents an Illumination Device on the Rover that supports nighttime driving and select science activities (Figure 9), derived from the example presented in [1]. This example was used as a proof of concept for developing the basic capabilities of the FMECA and FTA plugins, and it illustrates how the ontological concepts and their use in a model result in the creation of FMECA and fault tree artifacts that match traditional expectations.

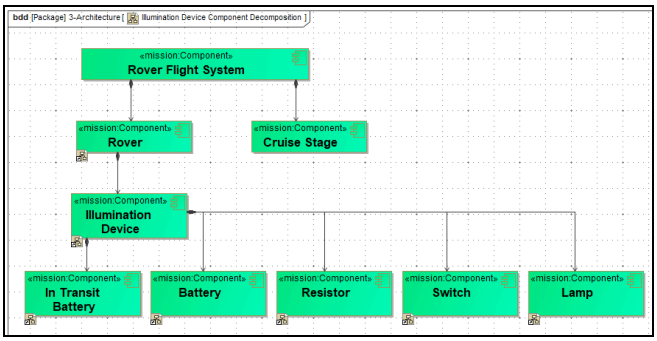


Figure 9. Illumination Device Model Architecture

In this subsection, we depict how the loss of the Battery results in losing the ability to provide a proper illumination, which in turn affects the performance of the Rover. The failure mode of interest (no electrical power supplied) is captured in the Battery behavior model shown in Figure 10 with a ViolationExplanation named "Battery No Output". Several states are explained by this ViolationExplanation, "Fully Discharged" and "Dead Battery". The ViolationExplanation indicates that these states are problematic with respect to the battery functionality.

Figure 11 describes the fault propagation path from the Battery component state machine to the Rover state machine, through the Illumination Device. The CauseExplanation "Dead Battery" explains the Battery state "Dead Battery" and analyzes the Illumination Device state "Permanent Insufficient Illumination", indicating that the battery state adversely affects the variable of interest at the Illumination Device level, i.e., its luminous flux. Then the CauseExplanation "Loss of Nighttime Driving Illumination" explains the Illumination Device state "Permanent Insufficient Illumination" and analyzes the Rover state "Nighttime Driving Unavailable", completing the propagation chain by indicating that nighttime driving is compromised. Note that the associations shown between the CauseExplanations and ViolationExplanations to the State Machines are abstractions of the actual dependency relationships that are established between these elements and the states owned by the State Machine, for visualization purposes.

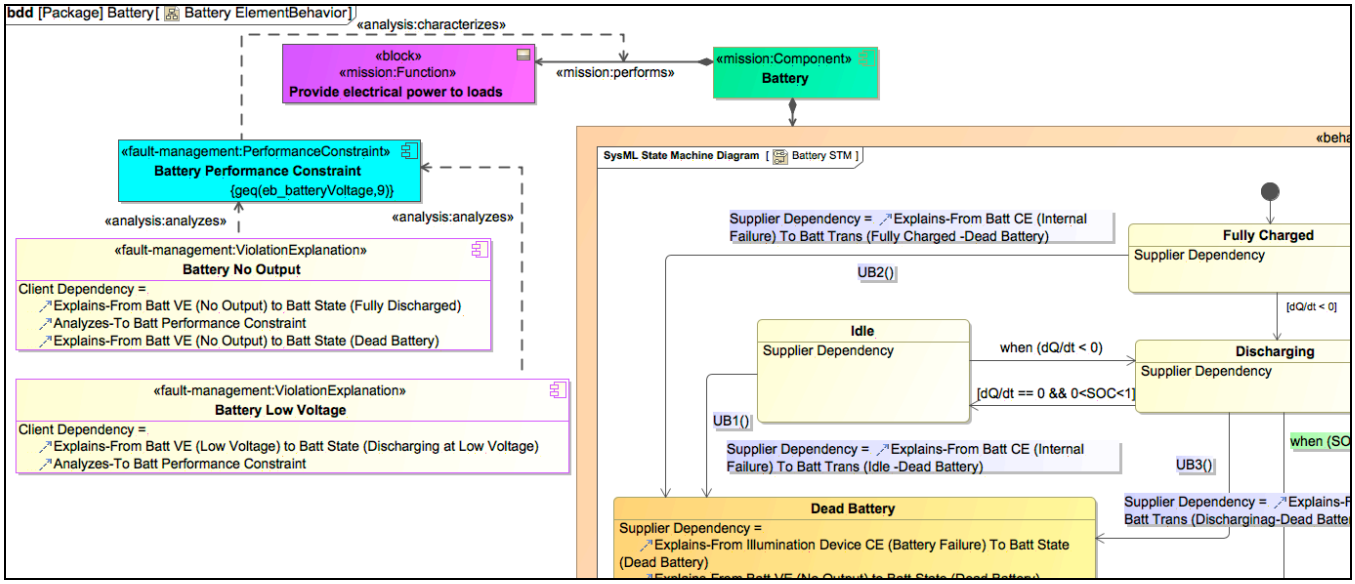


Figure 10. Battery Behavior Model Showing Battery No Output ViolationExplanation

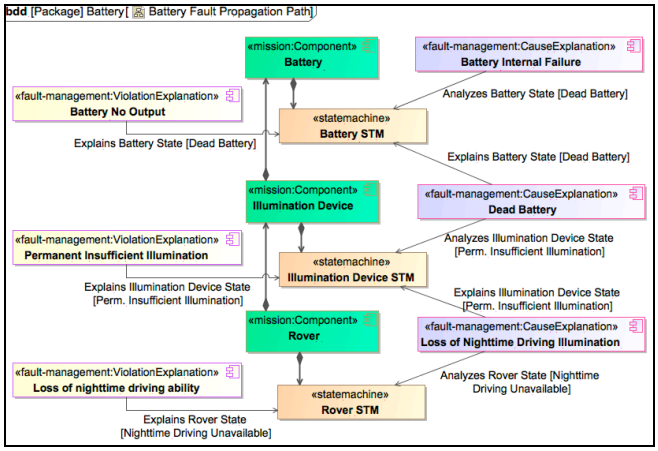


Figure 11. Fault Propagation Path Description for failure mode "Battery No Output" – *Associations from Explanations for visualization purposes only*

Figure 12 shows an excerpt of the FMECA plugin output in the modeling environment. The Rover was selected as the end effect level, and the highlighted row relates to the "Battery No Output" failure mode discussed above, presented in the "summary view". As expected in a traditional FMECA, the failure mode is properly listed, with cause and effect information. Here, the local effect is the "Permanent Insufficient Illumination" and the end effect is the "Loss of nighttime driving ability", inferred by the tool by traversing the failure propagation chain described above. Several columns to the right are not shown in the figure, containing information about detection mechanism, mitigation, likelihood and criticality rating. As described earlier in the paper, the "detailed view" (available through a tab) displays individual failure propagation paths. The "error summary" tab list issues, if any, detected by the plugin. Also, several csv and Excel files were created when the plugin ran the analysis.

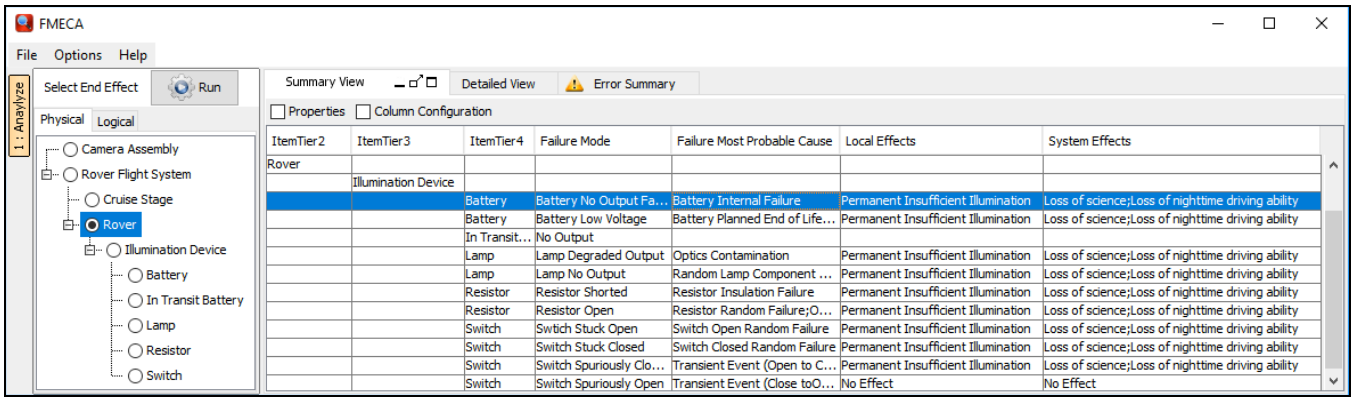


Figure 12. Result of FMECA Output for "Battery No Output" Failure Mode

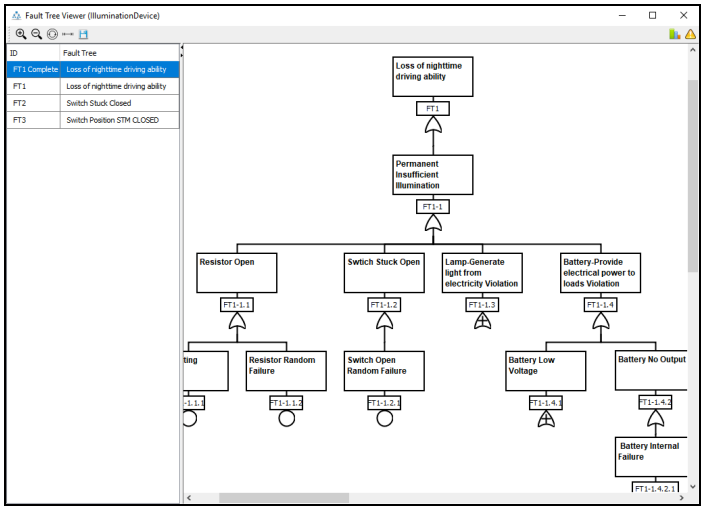


Figure 13. "Loss of driving ability" Fault Tree Output

Figure 13 depicts the Fault Tree analysis output in the modeling environment for the "Loss of nighttime driving ability" event, depicted as a ViolationExplanation in Figure 11. "Loss of nighttime driving ability" is the top event, with "Permanent Insufficient Illumination" shown as an immediate cause. Note that the "Battery No Output" event on the bottom right corner of the figure is not listed immediately below: as shown in Figure 6, a functional grouping was detected by the plugin, and it created an intermediate event to gather several failure modes related to the battery functionality. As seen in the fault tree, another battery state ("Battery Low Voltage") can lead to insufficient illumination, and it is grouped with the "Battery No Output" event under the plugin-inferred intermediate event. Several trees were created, as seen in the left panel in the FTA plugin interface, as repeating sections of the fault tree were detected by the FTA plugin. Transfer events were created as a result. Note also that in Figure 13, several branches were collapsed for readability, marked by a "+" symbol embedded in the gate.

Habitable Spacecraft Example

The Habitable Spacecraft model is a simplified representation of a NASA spacecraft architecture that was used to apply the fault management methodology to space systems, as well as to demonstrate the implementation of more complex modeling aspects, such as redundancy, cross tie application and multiple subsystems. The Habitable Spacecraft is a Vehicle with three subsystems: Power, Environment and Life Support System (ECLSS) and the Command & Data Handling (CD&H) System (Figure 14). In this subsection, we depict how the loss of the cabin air circulation results in failure of the life support system, which then affects the success of the mission. We then provide the FMECA and FTA output from the plugins tools.

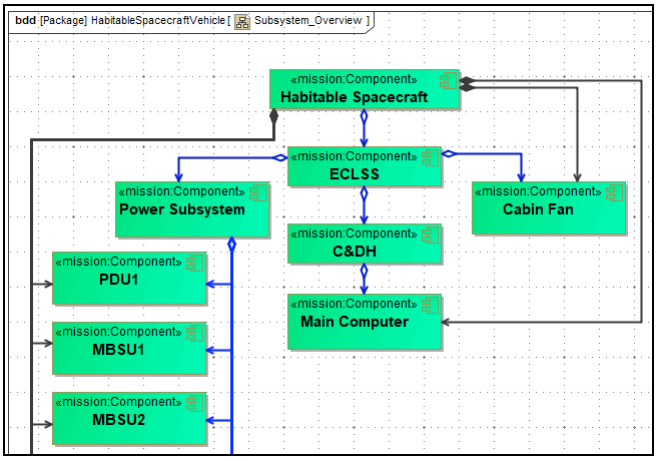


Figure 14. Fan in the Can Model Architecture

Figure 15 depicts the Cabin Fan behavior model. The Cabin Fan has the function "Circulate Air", and the failure mode of interest here is the "Loss of Cabin Air Circulation" captured by the appropriate ViolationExplanation that explains the state "Cabin Fan Failed off".

Figure 16 describes the fault propagation path from the Cabin Fan component state machine to the Habitable Spacecraft state machine, through the ECLSS subsystem. The CauseExplanation "ECLSS Failed Off", explains the Cabin Fan state "Cabin Fan Failed Off" and analyzes the ECLSS state "ECLSS Failed Off". Then the CauseExplanation "Habitable Spacecraft Failed Off" explains the ECLSS state "ECLSS Failed Off" and analyzes the Habitable Spacecraft state "Habitable Spacecraft Failed Off".

The FMECA result shows that the failure mode "Loss of Cabin Fan Air Circulation" has a Local Effect of "Loss of Life Support" and an end effect of "Loss of Crew" (Figure 17). Since the Cabin Fan is referenced by ECLSS, the ViolationExplanation, "Loss of Life Support", is reported as the local effect. A similar relation exists between ECLSS and the Habitable Spacecraft, and "Loss of Crew" is reported as the end effect. Since the CauseExplanation "Cabin Fan Malfunction 1" analyzes the failed state, "Cabin Fan Failed Off", the CauseExplanation is reported in the FMECA as a potential failure cause, and in the fault tree as a child event to the "Loss of Cabin Fan Air Circulation" event. Figure 18 depicts the Fault Tree analysis result for the "Loss of Crew" event. Note that "Loss of Cabin Fan Air Circulation" is depicted as an intermediate event. Note also that the fault tree has been collapsed under the power subsystem failure event, as indicated by the "+" signal below the event.

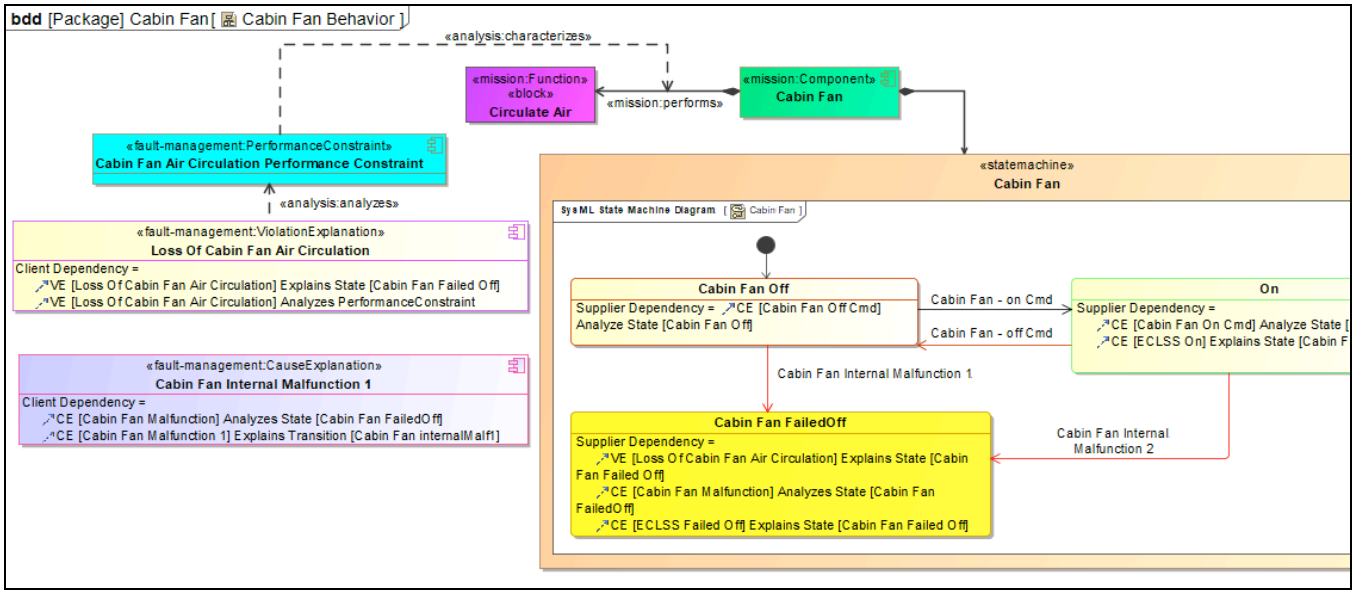


Figure 15. Cabin Fan Behavior Model

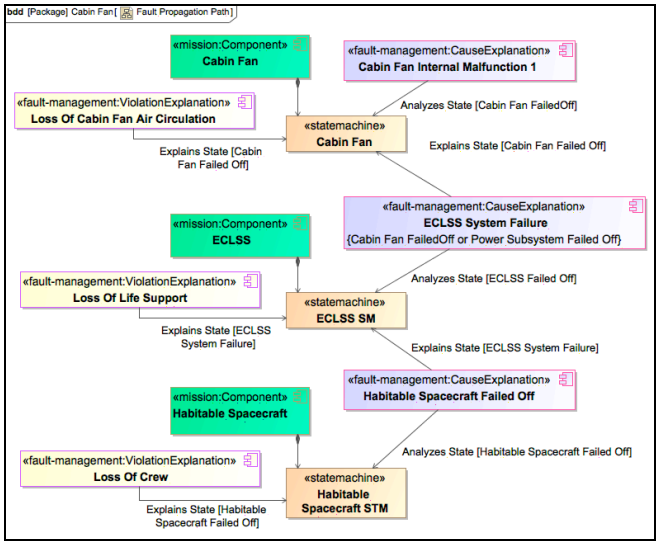


Figure 16. Fault Propagation Path Description for failure mode "Loss of Cabin Fan Air Circulation" – *Associations from Explanations for visualization purposes only*

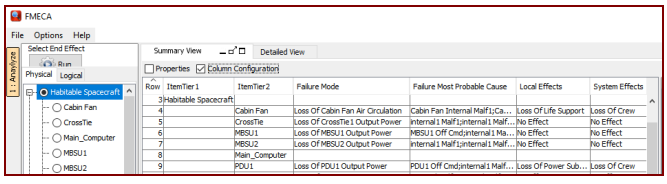


Figure 17. Results of FMECA Output for Loss of Cabin Fan failure and End Effects

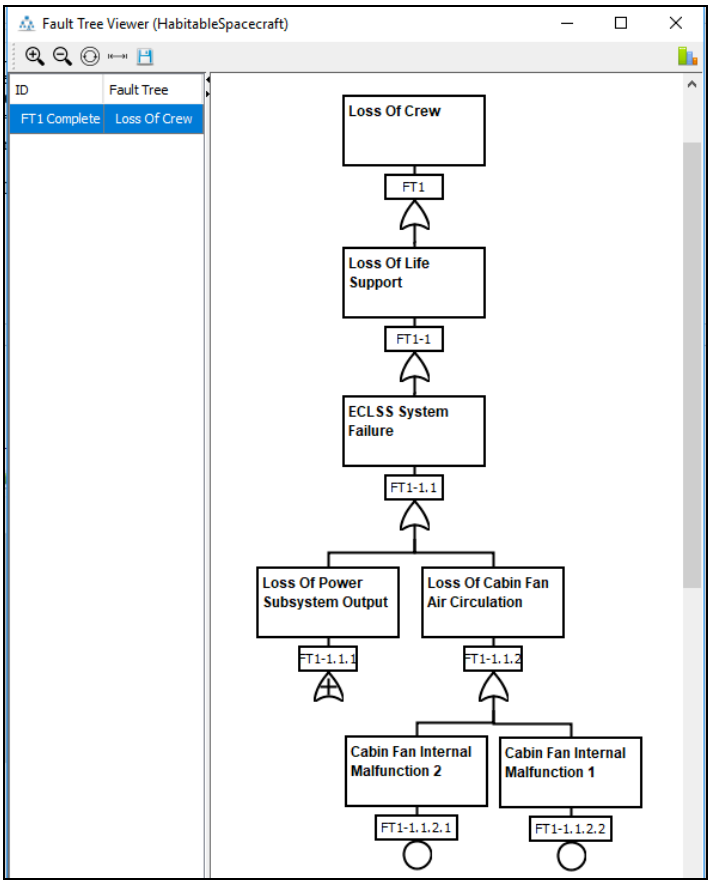


Figure 18. Fault Tree for "Loss of Crew" Event

5. CONCLUSION

This paper presented an approach to capture failure information in a modeling environment using the JPL-authored Fault Management ontology and a set of plugins designed to automatically extract two reliability artifacts, the FMECA and the Fault Tree. The mapping between the ontological elements and the artifact building blocks was described and illustrated through two examples.

The concept described in this paper demonstrates the ability to obtain products in a model-based environment that meet traditional expectations, as well as leverage model information to provide the user with in-depth analysis capabilities. This concept enables the integration of fault management early in the system engineering lifecycle, facilitating the discovery of design weaknesses and enhancing the capability to produce safe, hazard-free systems. This tool suite enables reliability engineers to use system models captured by system engineers to evaluate designs for potential faults, perform reliability analyses, and contribute to the overall system models by adding specific faults and associated reliability-related knowledge.

The FMECA and Fault Tree plugins have been used on projects at JPL, for example for the Soil Moisture Active Passive (SMAP) spacecraft or the Europa Clipper mission. Future work will include several extensions. For example, we will focus on expanding the Fault Management toolset to support reliability analysis for multiple Project Lifecycle Phases. For the FMECA and FTA tools, future additions will include features such as supporting active or passive redundancy, voting gates, and suggesting a wider range of criticality ratings to the user. Also, we will continue work on extending the plugins to support Probability Risk Assessment (PRA) and Bayesian probability theory with third-party tool support (e.g., MATLAB, Microsoft Bayesian Network (MSBNx)). Extensions such as these will help to analyze and mature more efficiently the design of systems for NASA and other industry applications.

ACKNOWLEDGEMENTS

The research was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration, as well as by Tietronix Software, Inc. under NASA/SBIR award No. NNX15CP03C. The authors want to acknowledge Dr. Howard Wagner, Ken McMurtrie, and Claus Nilson for their help in developing the toolset presented in this paper. The authors would like to thank Lui Wang from the Software Robotics Simulation Division of NASA Johnson Space Center for his support, as well as JPL's IMCE organization.

REFERENCES

[1] J.-F. Castet et al., "Fault Management Ontology and Modeling Patterns," in AIAA SPACE 2016, AIAA SPACE Forum, Long Beach, CA, 2016. doi: https://doi.org/10.2514/6.2016-5544.

[2] J.-F. Castet et al., "Ontology and Modeling Patterns for State-Based Behavior Representation," in AIAA Infotech @ Aerospace, AIAA SciTech Forum, Kissimmee, FA, 2015. doi: https://doi.org/10.2514/6.2015-1115.

[3] IEC 60812:2006, Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects Analysis (FMEA), International Electrotechnical Commission, 2006.

[4] MIL-STD-1629A, Procedure for Performing a Failure Mode, Effects, and Criticality Analysis, 1980 [Cancelled without replacement in 1998].

[5] Fault Tree Handbook with Aerospace Applications, National Aeronautics and Space Administration, 2002.

[6] Wang, L., Izygon, M., Okon, S., Garner, L., and Wagner, H., "Effort to Accelerate MBSE Adoption and Usage at JSC," AIAA Space 2016, p. 5542, 2016.

[7] Izygon, M., Wagner, H., Okon, S., Wang, L., Sargusingh, M.J., and Evans, J., "Facilitating R&M in Spaceflight Systems with MBSE," 2016 Reliability and Maintainability Symposium (RAMS), pp. 1-6. IEEE, 2016.

[8] John W. Evans, Frank J. Groen, Lui Wang, Rebekah Austin, Arthur Witulski, Steven L. Cornford, Martin Feather, and Nancy Lindsey. "Towards a Framework for Reliability and Safety Analysis of Complex Space Missions", 19th AIAA Non-Deterministic Approaches Conference, AIAA SciTech Forum, (AIAA 2017-1099).

[9] Sargusingh, M. J., Okon, S., and Callahan, M. R., "Cascade Distillation System Design for Safety and Mission Assurance," 45th International Conference on Environmental Systems, 2015.

[10] SAPHIRE – System Analysis Programs for Hands-on Integrated Reliability Evaluations, Idaho National Laboratory, https://saphire.inl.gov/.

BIOGRAPHY

Jean-Francois Castet *is a Systems Engineer in the Autonomy and Fault Protection group at the Jet Propulsion Laboratory (JPL), and he is part of the Flight System Engineering Team on the Europa Clipper Project. He is also involved in institutional activities to define modeling patterns for system behaviors, as well as infuse MBSE techniques into the fault management discipline. He received a M.S. degree from SUPAERO (Toulouse, France), and M.S. and Ph.D. in Aerospace Engineering from the Georgia Institute of Technology (Atlanta, GA).*

Magdy Bareh *is a Senior Flight Systems Engineer at JPL, currently working on the Mars2020 project. Most recently he was the supervisor for the Autonomy and Fault protection group. In his career, he has worked on flight system development and operations of several missions including Mars Science Laboratory, Spitzer Space Telescope, Dawn development, Deep Space 1, and Galileo operations. His areas of expertise include flight system design, Autonomous Fault Protection designs, avionics design and development, Verification and Validation campaigns and Spacecraft operations leadership. Magdy has a BS and MS Degree in Electrical and Computer Engineering from California State Polytechnic University, Pomona.*

Jeffery Nunes *is a Principal Engineer at JPL and serves as the JPL Systems Reliability Technical lead in the Reliability Engineering & Mission Environmental Assurance Section. He has 32 years of experience at JPL supporting many flight projects and institutional tasks, including the MBSE effort in the Office of Safety & Mission Success. He also serves as the JPL Reliability Technical Discipline representative to NASA.*

Michel Izygon, Ph.D. *is the CTO of Tietronix Software, a company specializing in advanced software technologies applied to Aerospace, Medical Devices and renewable energy power plants. Dr. Izygon has over 25 years of experience in software engineering including all phases of Software Development Lifecycle. He has extensive experience in UML, which he taught at the University of Houston – Clear Lake for ten years as well as in SysML. He is the Principal Investigator on the NASA SBIR research project focused on developing the Model Based Fault Management Engineering methodology and associated toolset.*

Shira Okon *is a Principal Engineer at Tietronix Software specializing in Model Based Systems Engineering. She holds an Aeronautical and Astronautical Engineering degree from Purdue University with an emphasis on Aerospace System Design. Ms. Okon has years of Fault Management expertise. She has training in CMMI and has obtained a Six Sigma Black Belt. She has used these skills to lead teams in identifying faults, analyzing root cause, and posing solutions for improvement. She is currently a SysML expert using MBSE methods on multiple NASA Engineering projects.*

Larry Garner *is a Senior Software Engineer at Tietronix Software specializing in Object Oriented Design, Software Integration, Systems Modeling (SysML) and Distributed Computing. Mr. Garner is a lead architect for MagicDraw plugin development and his efforts include tools that assist in model reasoning, data mining, model transformations, and integration with other external applications. Several of the tools developed by Mr. Garner are in use across multiple NASA centers. He holds a Bachelor of Science degree in Computer Science and Mathematics, and a Master of Science in Computer Information Systems.*

Emmy Chacko *is a Senior Software Engineer at Tietronix Software specializing in Object Oriented Design, Software Integration, Systems Modeling (SysML) and Distributed Computing. Ms. Chacko is a software engineer whose efforts include tools that assist in model reasoning, model transformations, and integration with other external applications. Prior to working at Tietronix, Ms. Chacko worked for Symantec where she assisted in developing the Norton Antivirus version 7.x. She holds Bachelor of Science degree in Electronics, and a Master of Science in Computers.*