Automated Incremental Design FMEA[1]

David R. Throop
The Boeing Company
2100 Space Park Drive
Houston TX 77058
281-483-5396
david.r.throop@boeing.com

Jane T. Malin
NASA Johnson Space Center
2101 NASA Road 1, ER2
Houston, TX 77058-3696
281-483-2046 and 281-483-2059
malin@jsc.nasa.gov

Land D. Fleming
Hernandez Engineering, Inc.
17625 El Camino Real, Suite 200
Houston, TX 77058
281-483-2058
land.d.fleming1@jsc.nasa.gov

Abstract–Failure modes and effects analysis (FMEA) is typically a costly manual process. We present the EPOCH (Engineering Product and Operations Cross-cutting Hybrid) Simulation for Failure Analysis software, which automates generation of FMEA from design models. The tool performs sets of scenario-based analyses, using the CONFIG hybrid discrete event simulator, to generate reports summarizing detailing analysis results. The paper describes how the tool uses this simulator, and how time-step modeling has be extended to handle failure cases that violate steady state assumptions and approximations that are well founded for the nominal case. The automation supports incremental FMEA: reporting how a design change alters the presentation of the functional effects of failures, as seen over a set of operational scenarios. We describe the representations of failure modes, scenario scripts and functional labels that supports the capabilities of this tool. An example is presented, based on analysis of a propellant production plant for a planetary base.

TABLE OF CONTENTS

1. INTRODUCTION

With EPOCH, we address some current limitations of FMEA–issues of making FMEA practical, reusable, and less labor intensive, and limitations of coverage and consistency.

FMEA in Current Practice

In contemporary practice for space systems, FMEA is a costly manual process. Typically, the design engineers write up one worksheet listing each failure mode. The reporting is to the 'box level,' where the box is an Orbital Replaceable Unit (ORU) or a similar assembly level–essentially, to the level where any failed unit can be pulled and replaced. The engineers list all the failure modes they consider relevant to the box's operation, and describe the effects (functional losses, sensor indications) at the box's interface. They choose descriptive but unstandardized names for the failure modes and for the effects. (As a typical example, a broken wire might be labeled either "Loss of Current" or "Loss of Voltage.")

For the Effects Analysis, a group of engineers investigates a system (or a subsystem)–usually from 5 to 30 ORUs. They look at each failure mode in turn, and describe how the failure effects propagate to the other system ORUs. They seek generally to show that each failure mode gives some off-nominal sensor signature (diagnosability analysis,) that the signature is unique (isolatability analysis) and to show that there is redundancy or an operations path to restore the lost function (recovery analysis.)

This approach leads to several problems. The analyses are labor intensive. The work product, the FMEA report, is not as standardized as other downstream applications require. System failures, which may occur without any component failure (e.g., sneak circuits,) are omitted. There is little reuse of earlier analyses of similar equipment. The analysis is performed once, fairly late in design. If the design changes after the FMEA (or because of it,) the FMEA is usually not repeated.

Generating FMEA Automatically–Much of the FMEA process can be automated. There are existing, commercial products for electrical-domain FMEA in the automotive industry. In EPOCH, we seek to adapt and extend these methods to the fluid-handling domain for space vehicles and equipment.

2. BACKGROUND

CONFIG Base and Propellant System Model

We use the CONFIG hybrid modeling and simulation environment to develop and run the automated test cases that produce the FMEA reports [1], [2]. CONFIG supports fast scenario-based simulation of systems in operation. It has been used to perform dynamic system-level validation of advanced control software for the Lunar Mars Life Support Phase III Test at Johnson Space Center, where four crewmembers lived in a closed chamber for 90 days with recycling of air and water [3]. The advanced autonomous agent software

[1] U.S. Government work not protected by U.S. copyright.

augmented low level control with a flexible executive and a planner, to manage storage and transfer of gases among chambers in the test. CONFIG is currently being used to gain early understanding of the in-situ propellant production (ISPP) system designs for Mars missions, and to explore alternative redundancy designs that would use advanced software for control and fault management. In Figure 1, the graphical interface to the CONFIG environment shows a model for an ISPP design. This ISPP design includes a CO_2 acquisition system that uses freezer technology (cc02), a Sabatier reactor for conversion of CO_2 and H_2 to methane and water, an electrolysis based O_2 generation system (H_2O-CELLS) and a zirconia-cell based system (Z-Cell) for generation of O_2 and carbon monoxide. The system includes cryogenic storage of liquefied gases.

For EPOCH, CONFIG models and simulations need to represent not only nominal operations and processes, but also the effects of failures and other problems. Component failure can produce problem inputs elsewhere in the system. These result in cascades of failures and off-nominal component states. Analysis should be possible when detailed performance data is unknown.

Discrete changes in behavior can result from certain types of failures and problem inputs that trigger a sudden change. These can change the control regime, the system configuration, or the capacity or performance of a system component. For example, discrete changes can result from bursts, shorts, errors or uncommanded actions. Changes in control, capacity or performance can also be continuous, gradual and nonlinear. These types of changes can result from buildups, wear, leaks and drifts. Some failures result in unchanging component states. Examples are components that are stuck, not adjustable or have no output. Some failures involve random variation in measurements or inputs. For any failure, the magnitude of the effect may be determined by the magnitude of the failure.

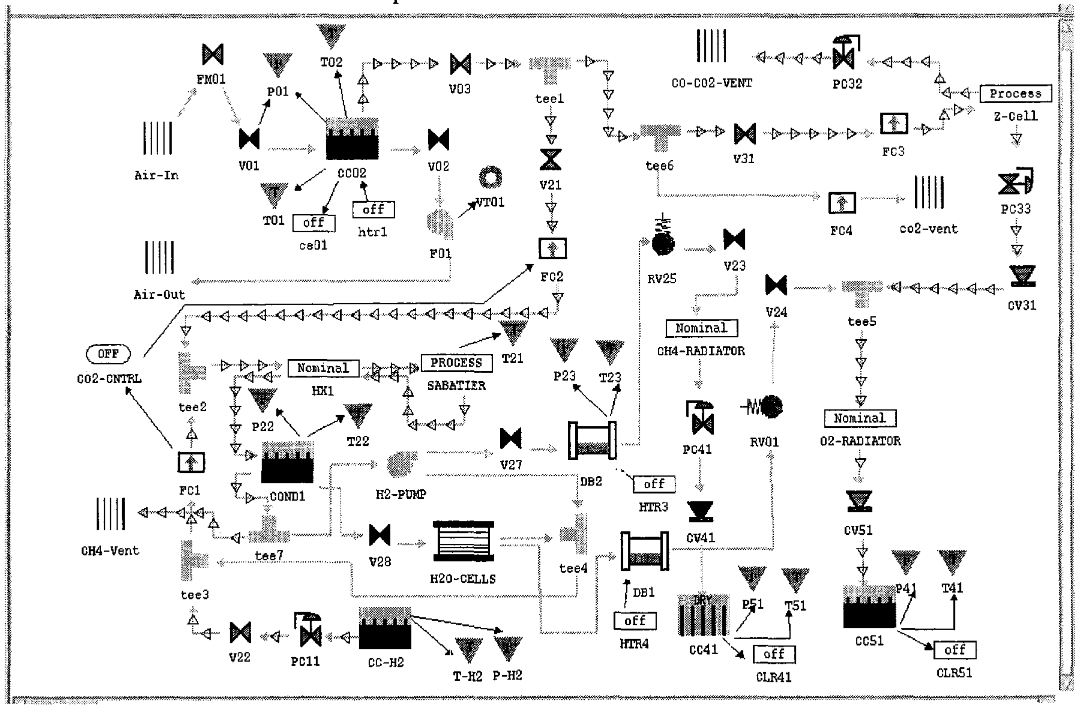


Figure 1. CONFIG Model of In-Situ Propellant Production System

The CONFIG hybrid discrete event / continuous simulator has been designed to meet these simulation needs. The hybrid models support approximation and simulation of discrete change, continuous dynamics and nonlinear and nontemporal relations. The hybrid models represent both discrete changes (commands, and things that happen over very short times, such as solenoids closing) and the continuous dynamics of tanks filling and temperatures changing. The discrete event basis gives a simple interface to operations scenarios, and to standard methods for stochastic variation. CONFIG uses a discrete time-step approach much of the time, to synchronize changes throughout the system of connected local models.

Models are composed of coupled local models from object-oriented libraries. Components have multiple modes, with each mode having state equations to generate behaviors, and conditions governing mode changes. Object-oriented model types for devices support modeling of components. Object-oriented activities support modeling of controllers, human operator procedures, actions and schedules. Modular coupled local component models with multiple behavior modes support dynamic changes in system configurations and operating modes of components.

CONFIG simulates complex flow regimes – multi-component mixtures, mixed-phase flows, variations in pressure, temperature and fluid density. Fluids are represented as first-class objects, with specialized methods for their behaviors. This is in contrast to the electrical domain, where the flow variables (voltage, current, power and resistance) may be taken to be state variables of the conductors. The same underlying facility supports computation for both fluid and electrical flow at the abstracted level of effort, flow and resistance.

In each simulation step, both local and global calculations produce the system behavior. Locally, the component behavior is calculated from its inputs and its state. Externally triggered and internally driven transitions result in changes in value assignments, with time delays.

Previous Work

In model-based reasoning in the past 10 years, there has been significant progress in qualitative and functional modeling. Initially, the dominant concept was that device function was an intrinsic characteristic of the behavior of the device, requiring a separate functional model from a qualitative behavioral model [4]. More recently, the dominant concept of function is as a role in a design, or as the indicators of that role. The indicators of function are selected effects of a device that are relevant to a purpose or service of the design [5], [6], [7]. These functional effects descriptions can be mapped to states in qualitative simulations of device behavior [8].

The concept of function as mapping to simulation effects has led to practical approaches for automated design analysis and FMEA. The FLAME system was developed to automate FMEA for automotive electrical systems [9], [10], and has since been commercialized as AutoSteve. Price and Taylor have also demonstrated an approach to constructing fault trees on the basis of the automated FMEA results. FLAME uses functional labels to abstract and interpret the results of sets of qualitative circuit simulations that test the operation of the system, with and without failure modes. Engineers identify failure modes of components and their effects on the system, and rank the severity, detectability and likelihood of the effects. FLAME models the components and their failure modes as simulation objects. Engineers select functions from a set of standard functional labels and match them to selected states in the simulation. The states are indicators that the functions are occurring. The functional label is mapped to a logical relation on a set of model variables.

FLAME models generate simulations of both nominal and failure behaviors, as the behavior of the circuit is exercised by changing switches and sensor states. In effects analysis, failure modes are imposed on the circuit during automated qualitative simulations. The functional labels interpret the simulated circuit behavior, to highlight the consequences of failures for system functions. The behaviors are reported in terms of functions, missing functions, malfunctions and sensor indications. The function differences between the nominal behavior and the off-nominal behaviors *are* the Effects Analysis.

We have pursued this approach to automated FMEA, by applying functional labels to hybrid quantitative simulation. Qualitative simulation was sufficient for FLAME's automotive electronics FMEA. But quantitative simulation is needed for analysis of space systems, in which there are subsystems handling fluids.

Chandrasekaran and Josephson [5] propose that functional descriptions also include the conditions and connections in the environment and system that are necessary for the device to achieve the intended effects. We have pursued this aspect of functional description, by including scripts for simulation scenarios as part of the specification for an automated FMEA. These scripts manage dynamic simulation scenarios that produce the sequence of conditions and configurations that are necessary for achieving functions in nominal operations. Scripts also manage the insertion of perturbations in failure scenarios. These dynamic simulation scenarios exhibit the complex interactions that are involved in achieving functions or producing undesirable failure effects. It is these complex interactions that analysts have difficulty understanding in manual FMEA.

3. EPOCH APPROACH

Goals and Domain Requirements

To meet the requirements of FMEA for space systems, EPOCH extends the approach and the representation developed in FLAME. We seek to represent both sudden failures and gradual degradations. Many failures in the space domain have variable magnitudes. A one cc/hr leak will have different symptoms than a one liter/sec event. We want to capture cascading system effects and interactions.

We are applying our models to the early design of the ISPP system for Mars missions. There is no detailed ISPP design yet. The incremental FMEA can be of particular use in trading off "what-if" design options early in the design sequence. We also seek to support follow-up analysis after design changes are accepted.

We also aim for process improvements; the functional labeling approach should yield more standard terminology for the FMEA report.

CONFIG interface–Our team used CONFIG as the EPOCH simulation engine. Its hybrid discrete event / continuous design supports reuse of the models. We are familiar with it and are able to modify and enhance it [1]. However, EPOCH is conceptually separate. We have endeavored to keep a clean interface between the two programs. The EPOCH scripts call out events as 4-tuples of *device : failure* *mode : magnitude : time.* The code that installs these events the simulation agenda must be specific to the simulator, but should be similar to any other discrete event simulation environment.

Scripts

EPOCH produces FMEA by generating system behaviors under simulation scenarios that mimic actual operation. The scenarios are encoded as a set of EPOCH *scripts* specifying, in several pieces, the simulation run. This includes:

1) The name of the script and the model to which it applies.
2) The initial conditions for the simulation (as initial variable values and component modes.)
3) Events to be placed on the event queue. Each event is realized as a variable or a mode-transition taking a value at specific time. These capture both expected operational actions and perturbations (failures or stresses.)
4) A set of values controlling the simulation, such as how long the time-steps will be.
5) A simulation HALT event.
6) Logging information (what variables and functions to log, when, to what file.)

Functional Labels

In a simulation, the trace of all the model variable values is the complete behavior for the simulated system. Such a trace is unwieldy. The *Functional Labeling* approach abstracts the interesting portions of behavior. For example, consider the case of a multiply-vented storage tank. It *Provides Tank Venting* as the OR of two instances of a lower level function, *Provides Valve Venting.* These are defined in turn as flow through each of two vent valves, the second with a slightly higher relief pressure. Compare the nominal case to a case with the primary vent-valve clogged. Although the valve-level functions will be different for the two cases, the tank-level function will be the similar – the tank is vented in both cases, but slightly later, with a slightly higher pressure. The effect of the primary valve closing can be reported in functional terms – a delay, a higher internal pressure – instead of as a trace of variable values.

CONFIG logs behaviors as traces of variable values. But this is too detailed for FMEA, which requires a abstraction from the numeric description to a functional description. For example, in multiply-vented tank, a trace of the flowrates through the two vent valves *implicitly* shows venting at the time when either flowrate goes positive. The FMEA needs to show how that function changes (is delayed, is not reached, occurs earlier) in the presence of faults. Realizing the function as a first-class object in the simulation facilitates that comparison. Further, by separating the function from the device variables, we will be able to compare the Venting function across different system designs–if we implement the system with a completely different kind of tank. This also allows the FMEA to express results that follow the naming used in the functional allocation.

EPOCH implements and detects malfunctions just as it does for functions–a function reached when it is not wanted nor expected is a malfunction, and a function that is never wanted or expected is always a malfunction.

EPOCH functional label objects include:

1) The function name.
2) A conditional expression defining when the function is achieved. This is a logical expression, composed of arithmetic and logical equations on
   a) Component variable values and modes. A function may be composed from values spanning multiple components. (The implementation allows a functional label to be defined once for a class of components and instantiated multiple times.)
   b) Sub-functions. (It is useful to be able to compose functions together, especially as this follows the decomposition in the functional allocation.)
3) A set of model-numbers that relate the conditional expressions to alternative designs. A function may be implemented differently in different designs. This conditionalization allows a single function definition to cover all the model alternatives.

Failure Modes

A failure mode is a property of a component class. Its effects are specific to an instance of that class. A failure will usually drive a component variable to an off nominal value. It may induce a mode change as well. Failures can occur spontaneously (from the viewpoint of the simulation model) or they may be induced. Some induced failures may reverse themselves if the inducing condition is removed. EPOCH failure modes include:

1) The failure mode name.
2) Its component class (which includes both a name and a part number.)
3) A precondition for inducing the failure. This encodes the set of conditions that will *force* the failure. It is not a condition necessary for the failure, as some failures may occur spontaneously.
4) The local effects. These are the effects at the component level (not necessarily the same as the effects for the Effects Analysis.) They are normally expressed as a component variable being driven to an off-nominal value, which may in turn trigger a mode change. Either of these may in turn fulfil the conditions for a function label. Failure modes may be instantiated with a magnitude. A gradual degradation failure mode will be expressed as a time series of incremental changes.

In addition to the above slots, which are used in the EPOCH analysis, there are others, taken from the FMEA worksheet, which are used for bookkeeping and for generating more informative FMEA reports:

5) The Verification and Validation information (worksheet name, date, author, sequence number, etc.)

6) The worksheet's explanatory text.
7) Mean time between failure figures for components.

The system *effects* in EPOCH's FMEAs are generated when a failure mode changes the occurrence of a system function.

The EPOCH Algorithm

EPOCH uses four nested cycles of simulation in generating FMEA.

The first cycle is the CONFIG simulation itself. A single script for the nominal case is run first. The model is initialized. Variables are set to initial values. Simulation starts. The log records the simulation-time when functions are (de)activated.

The second cycle iterates over the set of scenario scripts for the device. Each script is run in turn, and the functional behavior logged. These scripts capture the base scenario which may include operational sequences, human actions and errors, and failures and stresses on the system. The base scenarios do not include the individual failure modes injected in the FMEA generation, below.

The third cycle investigates the scenarios as perturbed by failure, iterating over the failure modes. At the initialization for the run, a triggering for the failure is added to the other script events placed on the event queue. The behavior (in the presence of each failure) is logged for each script. For many scripts, the failure will make no difference in the simulated behavior. Under other scripts, the failure *does* change *some* of the functions in the behavior (by changing the time when they are active, or by not activating them at all, etc.) The logs of the failure behavior are compared to the nominal case. The report of differences, in their raw form, is a machine-readable FMEA, to something like:

```
Device: H2O Storage Tank
Script: Initial Rapid Fill
Function: Venting
Failure: Inflow Crimp
Function occurs
  Base case: 23 sec
  Failure case: 278 sec
```

These reports are aggregated, filtered and phrased for a human reading.

Device	H_2O Storage Tank
Failure	Inflow Crimp
Function	Venting
Criticality	2A
Scripts	1) Initial Rapid Fill 2) Outflow Interrupted
Effect	Delayed

The functional reports are most useful when the criticality of the function is listed, and when the reports can be sorted by criticality figures.

The fourth cycle iterates over design increments. The machine-readable FMEAs are generated for alternative designs, and compared to the base case.

Exponential Flow Approximation

Adapting existing CONFIG simulation models to the simulation of failed conditions exposed some limitations. In particular, processes which *normally* quickly reach steady-state had been represented as discrete changes, moving to steady-state in a single time step. However, under some failure conditions, for those processes which slowed down the steady-state approximation became invalid. One approach to this problem would have been to use much shorter time steps in the simulation. But this would have made the simulations unworkably long. A better approach was developed to capture processes moving with exponential-decay towards their steady state values.

To illustrate this *exponential-flow* approach we present the derivation for one side of the ISPP's oxygen generation system. In it, liquid water is electrolyzed across a porous membrane. Oxygen is created on one side of the membrane and pressurizes a headspace. It flows out against the resistances in the network. In our development, we will treat the downstream network as a single resistance leading to a ground pressure. The network flow utility in CONFIG will likewise convert a complex flow network to a single resistance and ground pressure.

When the network resistance decreases suddenly, the oxygen outflow rises. The headspace pressure will then fall, forcing oxygen to flow more slowly through the outlet port. The headspace pressure will asymptotically approach a new steady-state pressure. Under normal operation, the time constant associated with this transition is much less than the time steps used in simulation. The transitory behavior of moving to the new steady state can be ignored–treating the oxygen outflow rate as the same as its production rate. But the steady-state approximation fails when the system's time constant is much larger than the timestep; the system goes through a significant period of increased oxygen outflow. Another approach is to say that the outflow rate is constant for the timestep. This works well for systems where the timestep is much less than the system time constant. But when the time constant is less than the timestep, this approximation results in the headspace being more than completely emptied in a single timestep.

Approximation method	***Timestep >> Time Constant***	***Timestep << Time Constant***
Steady State	Works well	Hides important behavior
Initial values	Overshoots, creates spurious behaviors	Works well
Exponential	Reduces to steady state	Reduces to initial values

For fluid flow systems, the exponential flow approximation works as follows.

1) Get the current contents (on mass or molar basis) and pressure in the component.
2) Calculate the current effective external resistance and ground pressure.
3) Under the assumption that the external resistance and ground pressure remain constant over the timestep, calculate
   a) The system time constant
   b) The steady state contents and pressure
4) Then, taking flow as decaying exponentially to the steady state value, calculate
   a) The contents and pressure at the end of the timestep.
   b) The net outflow over the timestep.
5) Using the external resistance and ground pressure, calculate the average pressure in the component necessary to drive the net outflow through during timestep.
6) Set the average pressure to be the value of the 'effort' variable visible to the CONFIG flow computation utility. The flows and potentials throughout the system are updated according to a linear circuit approximation.

4. Preliminary Results

EPOCH is still being implemented and tested. The first work done was on scripts and on the exponential flows. Figure 2 shows a two-tank system used to test the exponential flows approximation. It has a system time constant of two seconds. When simulated using the *Initial Values* approximation and with time steps larger than the time constant, the generated behavior shows a diverging oscillation. Too much material flows in a single timestep, the pressure difference between the two tanks more than reverses, and even more material flows back the next timestep.

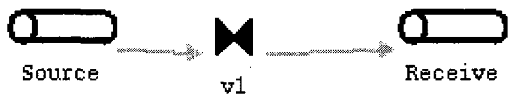


Figure 2. Two connected tanks

Figure 3 shows the trace of the first tank's pressure, simulated using the exponential flows approximation. For timesteps at or below the time constant, the pressure the trace closely matches those under the *Initial Values* approximation using small timesteps. For the larger timestep of 4 sec, the *Exponential Flows* trace shows a slower approach to equilibrium, but does not oscillate or diverge.

The scripts have been useful as a software testing tool. Exponential flows were implemented for one-, two- and three-port tanks. It was useful to have scripts in place that ran all the test models through a series of scenarios. These runs generated traces of whether (or how) their behaviors changed as the implementation was debugged.

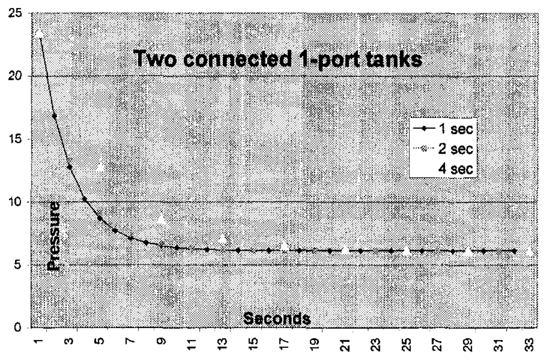


Figure 3. Pressure behavior in first of two connected tanks using exponential flow approximation, for selected timesteps.

5. Discussion

Scripts and Coverage

In traditional FMEA, engineers trace effects of how a failure propagates to adjacent or downstream components. There is a tacit question in this work, "Under *what conditions* does the propagation proceed?" and a related concern, "Are the conditions we have considered adequate to capture all the effects propagation at issue?"

EPOCH makes these conditions explicit, as the set of test scripts. This sharpens the latter concern: "Are the scripts adequate to test the propagation of failure effects?" We have no proof of completeness for a set of scripts, but we adapt current practice to provide a measurement of completeness. A minimal set of scripts exercises each function at least once, when run in the base case (with no failure present except ones in the script). But some functions (such as redundancy, warning, safing) are only exercised under failure. Some scripts will then include failures to trigger these functions, and the FMEAs of those scripts will show the effects of multiple failures.

This resembles the coverage issue for software testing. One yardstick for software testing is the extent to which code is exercised. The analogous measure for model testing is the extent to which the different device modes are exercised. It is not that a set of scripts must exercise every mode for every component; a valve may be installed under circumstances where no plausible script could cause backflow. But a logging of unexercised modes should force the engineer to confront the question: "Are you aware that you've never tested this backflow mode?" Such reporting is planned for EPOCH but not yet implemented.

Software testing tests code against a specification, but many accidents and major losses have been due to flaws in software requirements [11]. The specification can be incomplete because features of the software are not included in it, and it can be incorrect because system complexity and interactions are not accounted for in design. An analogous potential incompleteness weakness for EPOCH is the omission, from the analysis or the model, of "unintended" functions or malfunctions that are permitted by the devices in the system design.

Another potential weakness is incorrect specification of an operational scenario to achieve an intended function, based on flawed design understanding. This failure to achieve the function can be observed in the scenario-based simulation, and corrected. Another incorrectness weakness is failure to map functional labels to the functions that the designer intended. Consider a car with the kind of headlamps which swivel downward and are covered when not in use. If our *Provides Highbeam Light* functional label omitted a check on the headlamp's orientation, then the FMEA would show that a failure of the swivel mechanism would not interfere with *Provides Highbeam Light.*

Functional Labels

Functional labels share the same question of coverage and suitability: what is the *right* set of functional labels for a device? If the design is sufficiently mature, the functional allocation should give a nearly complete accounting of the device's functions. Adopting these as the functional labels assures a common functional terminology. Hopefully, the functional allocation will have named functions in a way that makes the FMEAs comprehensible.

The FMEA worksheets provide functional labels for those malfunctions that are not simply the negation of allocated functions. For example, *maintaining fluid purity* is not normally a function allocated to a piece of piping, but it may still have a malfunction of *contamination.*

In a qualitative simulator like FLAME, the named qualitative states often match the function of concern. In a numeric simulation, the functions must be mapped from numeric values to device states. The challenge is to keep taming the additional tasks for the engineer. Selecting function names from a previous electronic source (such as the functional allocation or FMEA worksheet) is a essential to keep the FMEA specification task manageable.

Models

The EPOCH work has driven insights about the FMEA process. Traditional FMEA does not capture the behavior of components when they are given off-nominal inputs. For instance, a FMEA worksheet for a pump captures its failures and its outputs under those failures. But the pump designer is not normally asked to document the pump's behavior and outputs when the input power is over voltage, or when its cooling fails, or when the input fluid is corrosive. Although the traditional FMEA process may call for this knowledge on an as-arises basis, it is not systematically generated.

Automating FMEA requires this knowledge. The component models need to respond to off-nominal inputs when the conditions arise during simulation. Including this knowledge can add significantly to the data gathering and model validation effort. However, successfully incorporating these behaviors greatly enhances reusability of component models. It also increases the sensitivity of the analysis to problem inputs and cascading effects.

6. CONCLUSIONS

FMEA can be automated for space systems, using numeric simulation, analyzing systems with complex fluid flows. Scripts of operational scenarios exhibit the complex interactions that are involved in achieving functions or producing undesirable failure effects. These scripts make FDIR coverage issues explicit. The failure effects report can be cast in terms meaningful to the engineer. Functional labels use the same terminology as the FDIR worksheets and the functional allocation. Incremental design FMEA is a straightforward extension of the automated FMEA approach. The change in the activation of these functional labels gives a high-level comparison between design alternatives. The list of device-modes not activated during FMEA generation is a key to completeness problems in the test scripts.

7. REFERENCES

[1] Malin, J. T., D. Ryan and L. Fleming, "CONFIG - Intelligent Modeling and Analysis of Systems and Operations," Proc. EXPERSYS-94: 6th Intl Conf on Artificial Intelligence and Expert Systems Applications, Gournay-sur-Marne, France: IITT-International, December 1995: 771-775.

[2] Malin, J. T. and Fleming, L. "Enhancing Discrete Event Simulation by Integrating Continuous Models". Working Notes of AAAI Spring Symposium on Hybrid Systems and AI, Stanford, CA, March 1999.

[3] Malin, J. T., Fleming, L. and Hatfield, T. R. "Interactive Simulation-Based Testing of Product Gas Transfer Integrated Monitoring and Control Software for the Lunar Mars Life Support Phase III Test." SAE Paper No. 981769. SAE 28th International Conference on Environmental Systems, Danvers MA, July 1998.

[4] Vescovi, M. Iwasaki, I, Fikes, R. and Chandrasekaran, B. CFRL: A language for specifying the causal functionality of engineered devices. Proc. Eleventh National Conference on Artificial Intelligence. AAAI Press/MIT Press, 1993: 626-633.

[5] Chandrasekaran, B. and Josephson, J. R. Representing function as effect: Assigning functions of objects in context and out. Working Notes of AAAI-96 Workshop: Modeling and Reasoning with Function, Portland, OR, August 1996: 30-37.

[6] Malin, J. T. and Ryan, D. "Device Roles in Operational Configurations". Working Notes, IJCAI-95 Workshop: Representing and Reasoning with Device Function, Montreal, Canada, August 1995: 90-91.

[7] Malin, J. T., and Schreckenghost, D. L. "Problems of Mapping between Functions and Device Phenomena". Working Notes, AAAI-96 Workshop: Modeling and Reasoning with Function, Portland, OR, August 1996:14-15.

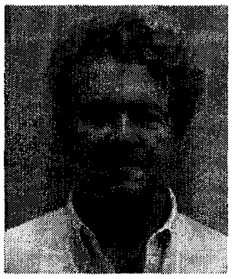
[8] Franke, D. W. Deriving and Using Descriptions of purpose. IEEE Expert. April 1991: 41-47.

[9] C. J. Price and N. S. Taylor, Multiple Fault Diagnosis Using FMEA, in Proceedings of AAAI97/IAAI97 Conference, Providence, Rhode Island, July 1997: 1052-1057.

[10] C. J. Price, Function Directed Electrical Design Analysis, Artificial Intelligence in Engineering 12(4), pp. 445-456, 1998.

[11]Leveson, N. *Safeware: System Safety and Computers*. Addison-Wesley 1995.

8. BIOGRAPHY



***David R. Throop** has been an Artificial Intelligence Specialist with The Boeing Company since 1992. He provides engineering software support in the Intelligent Systems Branch in the Automation, Robotics and Simulation Division in the Engineering Directorate at NASA Johnson Space Center. He oversaw development of FMEA modeling software and its use in the modeling of the International Space Station. He received a Ph.D. in Computer Science from the University of Texas in 1992, where his dissertation work was in Model Based Diagnosis. He received a Bachelors of Chemical Engineering from Georgia Tech in 1979.*

***Jane T. Malin** is Technical Assistant and Computer Engineer in the Intelligent Systems Branch in the Automation, Robotics and Simulation Division in the Engineering Directorate at NASA Johnson Space Center, where she has led artificial intelligence (AI) research projects for 15 years. She manages the Adjustable Autonomy Testbed project where she has developed designs, design principles and technology for intelligent systems for collaboration with human operators. She has led development of the CONFIG simulation tool for evaluating intelligent software for operation of space systems. She has led research on intelligent user interface and expert systems for real-time monitoring and fault management of space systems. She received a Ph.D. in Experimental Psychology, focusing on cognitive psychology, from the University of Michigan in 1973.*

Land D. Fleming is a Computer Systems Specialist working as a contractor in the NASA Johnson Space Center Automation, Robotics, and Simulation Division since 1990. He has been involved in both the development of computer simulation tools and their application to space systems. He received his M. S. degree in Computer Science from De Paul University, Chicago, Illinois in 1987.