08WATC-0023

Software FMEA Automated and as a Design Tool

Herbert Hecht and Rebecca Menes
SoHaR Inc.

ABSTRACT

We describe a method and tool (MOCET: Model based Certification Tool) that generate computer-aided software FMEA. The method and tool are used in verification and certification of avionics programs and have great benefits also in the design phase offering a safer, more complete design while reducing the costs of late stage discovery of safety issues. The use of a FMEA tool at the early stage of design simplifies and optimizes the implementation of Defense-in-Depth, a technique of multiple layers of protection to prevent and mitigate failures and accidents. MOCET's ability to aggregate all failure modes that produce a given high level effect provides an ideal vehicle for optimizing placement of broad spectrum recovery mechanisms during the design stage.

INTRODUCTION

In previous work we have discussed the application of the MOCET software FMEA generator in verification and certification of avionics systems [1]. The current paper shows how an extension of these capabilities can also be put to use in the design phase to produce a robust and certifiable design by:

- Identifying potential failure modes that can produce safety or mission critical failure effects
- Inserting effective detection provisions
- Recommending suitable means for protecting against the effects of the detected failures

The methodology described here is based on the use of MATLAB/Simulink or a similar model-based design tool that provides primitives with predictable failure modes [2]. MOCET combines an intelligent parser of model files, with sets of libraries that cover failure modes, local and end effects and detection and mitigation methods. The parser automatically translates a design model into a "part list" tree (similar to a hardware Bill Of Materials). The part list includes the hierarchical structure of the model (how parts associate into subsystems and systems). Once the hierarchical structure and part identification are generated the tool can automatically associate failures with each part. Based on the hierarchical structure generated by the parser, the tool can automatically identify local effects, and propagate these to the system level. In its use as an FMEA generator MOCET associates each failure mode with an end (system level) effect. When the severity level of this effect exceeds a user selected threshold, the added design tool features can recommend detection and mitigation measures to reduce the severity. The effectiveness of the tool is a function of the quality of the libraries. As an organizational tool, MOCET also performs as an organizational knowledge base, the libraries being a growing and living organism becoming more useful as the tool is used and new entries are saved in the failure, effects and mitigation/detection directories. The following section summarizes relevant features of MOCET as an FMEA generator. From this we then develop the extensions that make it into an effective tool for designing failure detection, circumvention and recovery techniques. Because design is not a completely deterministic activity, we find that the tool provides choices rather than prescribes a single provision.

MOCET AS AN FMEA GENERATOR

The purpose of an FMEA is to draw attention to failure modes that can cause catastrophic or mission impairing effects. These effects are usually identified by the system engineer on the basis of fault tree analysis and related techniques. For flight or engine control systems the most serious effects are typically loss of output, freezing of output, or un-commanded changes in output. By convention, these are designated as severity I effects. Somewhat less serious are low gain, sticky (lagging) output and restricted range of output; these increase the workload of the flight crew and may be designated as severity II. Lower severity failure modes include offsets from the desired attitude or flight path, drift in attitude or flight path, and limit cycling. Once these effects and the associated severities have been decided on, they are input as end effects data into MOCET.

The main part of the analysis consists of (a) importing the model structure from the Simulink file, (b) selecting failure modes suitable for each element and associating failure effects with these, and (c) determining whether the design provides failure detection and mitigation measures. Details of these activities are presented in the following paragraphs with the longitudinal flight control system shown in Figure 1 serving as an example. When the analyst selects the file for this control system MOCET returns the screen shown in Figure 2. The element designation and identification numbers are automatically generated from the Simulink file by the MOCET parser.

The listing is in function and format equivalent to a bill of materials (BOM) that is conventionally used as the starting point for hardware FMEA. This gives the MOCET generated FMEA a much greater claim to completeness than one that is generated from a functional partitioning of the software. The definition of functions is subjective; specifically, exception handling may not be separated from the primary function that it serves. Listing of failure modes according to designated function may miss failures associated with sub-roles such as exception handling. Since many software failures are due to faulty exception handling [8, 9] the suppression of it in the FMEA worksheets may lead to incomplete safety assessment.

To further populate this screen the analyst selects a row and right clicks which causes a failure modes menu to drop down that is tailored to the specifics of the highlighted block. This is shown in Figure 3. The local failure effect is usually (but not always) uniquely associated with the failure mode. The analyst selects failure modes and effects that are applicable to the highlighted row. A local failure effect different from that shown on the screen can be substituted by editing.

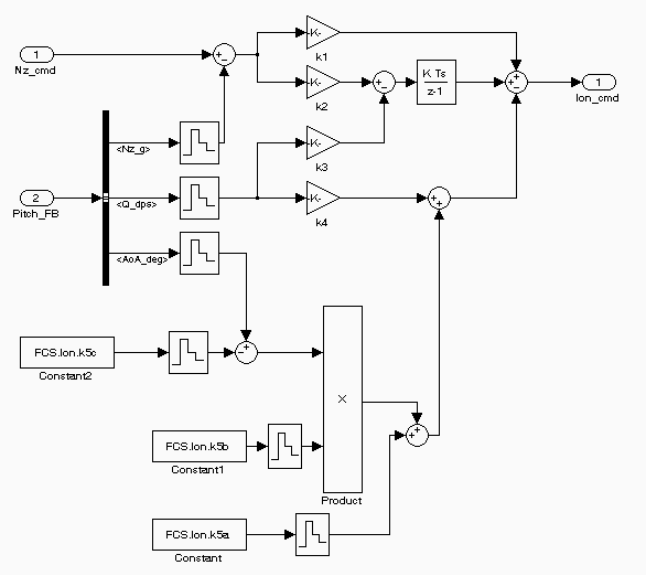

Figure 1. Longitudinal control system

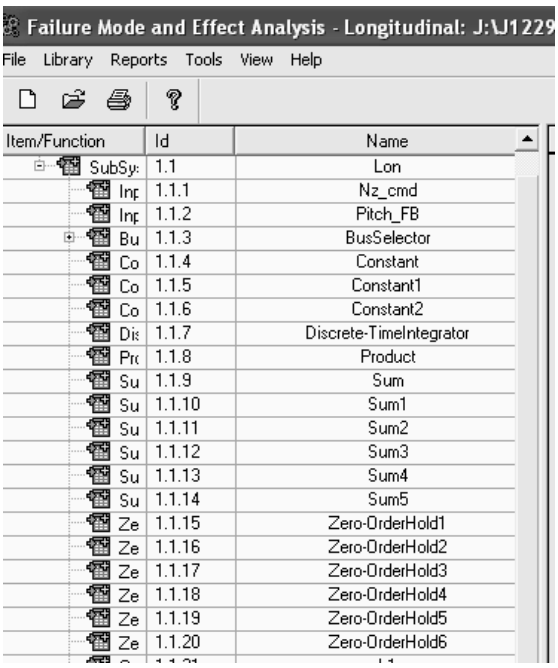

Figure 2 Initial MOCET screen for system of Figure 1

Further right-clicking then causes another menu to drop down from which the end effect and severity are selected as shown in Figure 4.

To protect against the highest severity failure effects the design should include failure detection and recovery (mitigation) provisions. To establish the existence of these the analyst (usually working with the system engineer) must review the design. The provisions that are applicable are then entered into another drop-down menu as shown in Figure 5. At present these actions are not automated but based on techniques similar to those described below in the Implementation section they will be automated in future versions of MOCET.

SAE ARP4761 [7] defines the FMEA but does not mention specifically the application to software. However, the objectives for FMEA in that document are applicable to our discussion. It is of interest to show how ARP4761 may be used to guide the effort. The FMEA furnished by MOCET is decidedly of the piece-part type (App. G-3.2.2), with the Simulink blocks replacing hardware piece parts. There are minor deviations in format; e. g., MOCET provides no column for phase of operation since software is typically analyzed separately for each phase. Also, failure mitigation provisions (redundancy, error correcting code, etc.) are included in the MOCET worksheets but are not shown in the App. G worksheets. Software is heavily involved in the implementation of mitigation provisions and the analyst must be informed about them. As will be discussed later, MOCET can play an important part in identifying suitable mitigation provisions where they were not present in the original design. MOCET records failure effects at several levels of the hierarchy (local, next higher and system level) and thus does not need the Failure Modes

and Effects Summary (FMES) described in Appendix H of ARP4761. However, if the overall safety assessment process requires that an FMES be furnished, this can easily be generated from the data fields of the MOCET worksheets.

INTERACTION OF DESIGN AND CERTIFICATION

The preceding section showed the use of MOCET as an analysis tool in support of one of the milestones leading to certification. We now turn to the proactive use of the tool during design of equipment that is intended to be certified for compliance with AC 25.1309-1A [3]. The AC requires that the occurrence of malfunctions that can impede "…continued safe flight and landing" shall be extremely unlikely, defined as having a probability of occurrence of no more than 10-9 per flight-hour. Also, malfunctions that impose a significantly higher than normal workload on the pilot must be controlled to a probability of less than 10-5 per flight-hour. Thus, an

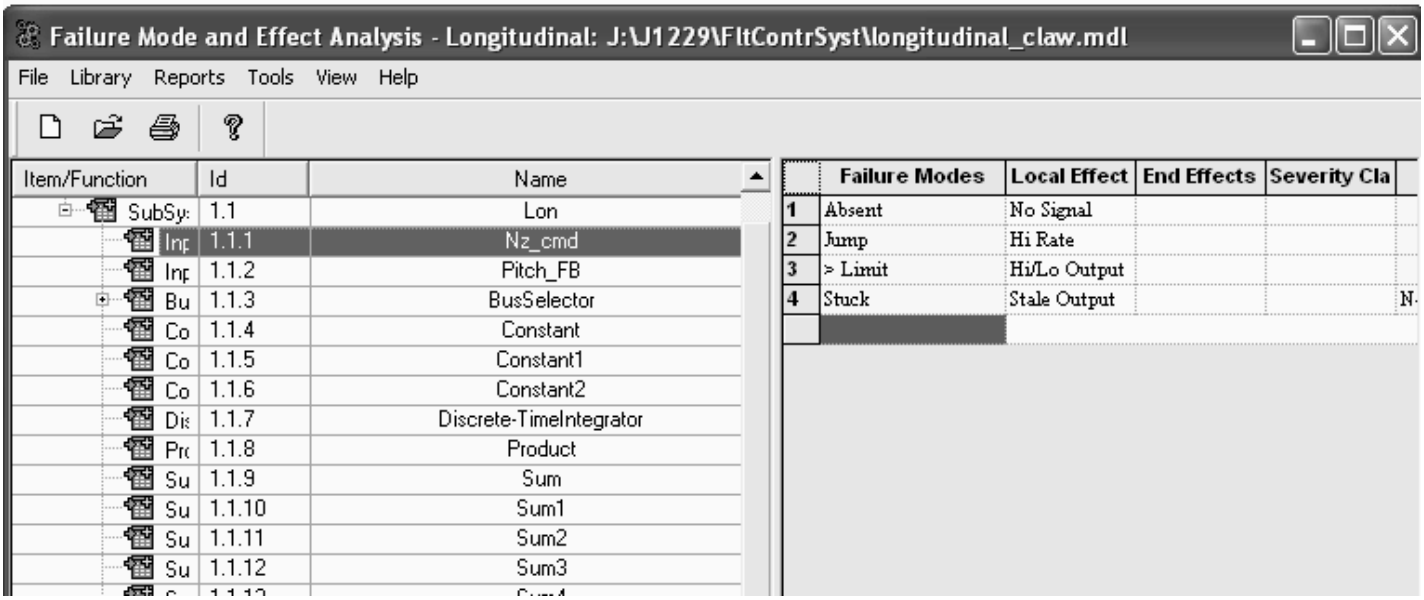

Figure 3 Failure Mode Selection path.

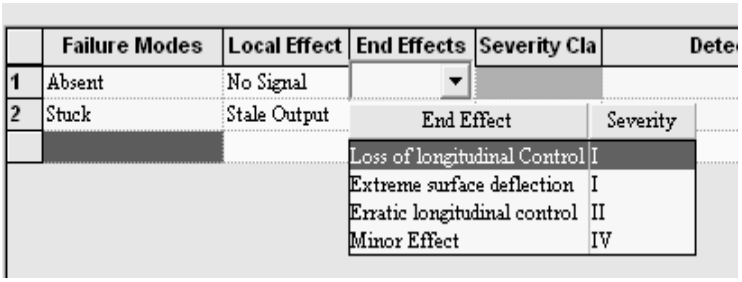

Figure 4 Selecting end effects and severity

important clue to the designer is a listing of where such malfunctions can arise.

The use of an FMEA tool at the early stage of design reduces the cost of late staged discovery of safety issues and it simplifies the implementation of Defense-in-Depth (multiple mitigation measures) as will be discussed later. The ability of MOCET to aggregate all failure modes that produce a given high level effect provides an ideal vehicle for optimizing placement of broad spectrum recovery mechanisms during the design stage. As an example, where a failure in the processing of an adaptive elevator command loop in a flight control system can cause extreme surface deflections (system effect: structural damage or loss of aircraft) it is possible to design a non-adaptive controller to take over when extreme surface deflections are commanded. However, extreme surface deflections can also be caused by other failures for which substitution of the non-adaptive controller would not help with recovery. A broader spectrum recovery mechanism will therefore be more economical and reliable (each added recovery path needs to be analyzed and tested).

As an added safety provision (or in some cases as an alternate) MOCET can be tasked to list all failures that result in a high rate of attitude change. Where the high attitude rate is caused by a software problem, an effective mitigation is switching to an alternate program; but again, a high rate of attitude change may result from some failures outside of the elevator command loop, and recovery provisions for these failures can therefore be combined with (or substituted) for those described above.

Less catastrophic failures may result in slow attitude changes that may not be detected at that level but that will result in a detectable deviation from the desired flight. Thus, additional broad coverage failure protection may be provided at the flight path level. Broad detection coverage is not desirable in many cases because it does not point to a specific cause, leads to later detection than specific provisions, and because of this permits proliferations of faulty data that may persist even after

corrective measures for the original failure have been completed. Therefore when a MOCET listing indicates that a failure is detectable at a high hierarchical level it does not necessarily mean that lower level failure detection can be deferred. Engineering judgment is still required to select appropriate detection and mitigation provisions. But the listings provided by the tool facilitate the accomplishment of a robust design and assure that no critical failure mode is overlooked. Designing multiple detection and mitigation provisions for a given class of malfunctions or security concerns is referred to as *defense-in-depth* [4] because of its similarity with military doctrine for defending an important position by multiple defensive perimeters (mine fields, trenches, and block houses), and an example of this technique for the longitudinal channel of a flight control system is now discussed.

A typical initiator of a serious longitudinal axis failure is a spurious high rate elevator command. Such a command can be detected, e. g., at the computer output, but the detection level must be set high enough to allow rapid elevator motions that may be required for obstacle avoidance. Most abrupt command changes can be prevented by this means of detection but probably not all. Those that escape detection will result in a high pitch rate, and this parameter can be used as a second line of defense. Because commanded pitch rate is usually available within the flight control data base, a criterion for failure detection can be the difference between commanded and actual pitch rate; this detection level can be set low and without impairing aircraft maneuverability because desired high rates of attitude change will be reflected in the commanded rate. A third line of defense can be designed by noting deviations from the commanded pitch attitude or the desired flight path. MOCET as a design tool will show all these options and provide assistance with their implementation as described below.

IMPLEMENTATION

There is an extremely large number of possible system and software failure modes for which the design should provide protection. But the designer can be directed to suitable detection and mitigation provisions by considering generic failure effects at a functional level as shown as a partial example in Table 1.

The effects are ascribed to hardware components but they are also applicable to software that processes or is associated with the listed hardware output.

When a signal with a typical noise content that changes the two or three least significant bits every computer cycle remains unchanged for several (n-) cycles this indicates with high probability that a failure has occurred in one of the elements of its channel. The condition is detected by subtracting the current signal value from that of each of the previous n-cycles. The diagnosis of a stuck output is made when all of the differences are zero. An example of the Simulink implementation of a 3-cycle wait is shown in Figure 6. The assertion element initiates the corrective actions. The selection of n- depends on the expected noise content of the signal and the tolerable delay for furnishing an alternative.

When a complete output failure can be detected by other means, such as the heart-beat shown for single processor in Table 1, it is preferable because the use of the n-cycle wait always involves a delay. If smart sensors are available, they generally detect output failures earlier than the n-cycle wait and are therefore preferable

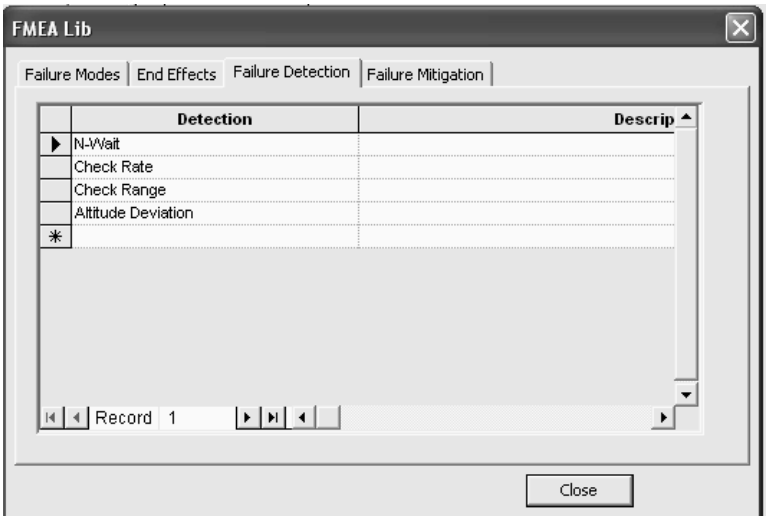

Figure 5 Selecting detection provisions

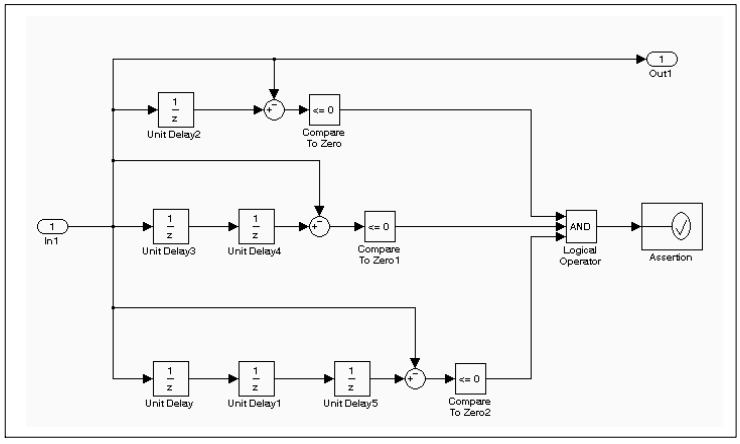

Figure 6 Implementation of 3-cycle wait

Table 1 Generic Failure effects

Function	Failure Effect	Detection	Compen-sation	Re-marks
Sensor - single	Zero output	n-cycle wait	Analytical redundancy	
	Full scale	Range	"	
	Jump	Range rate	"	
Sensor – dual	All	Comparison	Delete failed one	Note 1
Sensor – triple	All	Voting	Median value	
Processor - single	No output	Heartbeat	Safety mode	Note 2
	Stuck	n-cycle wait	"	
	Extreme output	Range	"	
Processor – dual	All	Comparison	Delete failed one	Note 1
Processor – triple	All	Voting	Majority vote	
Memory - single	No output	Host processor	Safety mode	
	Output error	Error det. Code	Repetition	Note 3
		EDAC	Cotinue or repeat	
Memory – dual	All	Comparison	Delete failed one	Use error det. code
Memory – triple	All	Voting	Majority value	
Bus	Same as processor			
Output element	Stuck	Force feedback	Safety mode	
	Broken/open	Outer loop	"	
Comparator	No output	Heartbeat	Single mode	Use current active
	False positive	Self-test	"	"
	False negative	External comp.	"	"

Notes:
1. The failed unit may be identified by (a) self-test, (b) deviation from last accepted value, (c) magnitude of output
2. The detection processing must be assigned to a mission computer or a specialized diagnostic processor
3. When repetition does not clear the problem use safety mode

CONCLUSION

The use of model-based software development facilitates the assignment of failure modes to each model element and thereby makes possible the systematic review of the failure modes. A developer who uses MATLAB/Simulink can thus employ MOCET to automate many steps of the software FMEA generation. This not only reduces the effort but also eliminates many sources of error and goes a long way in demonstrating that the FMEA is complete.

Based on these capabilities of MOCET, we have now added features that permit the designer to prepare for the FMEA by letting the tool recommend failure detection and mitigation provisions. This represents a further step in reducing design effort, eliminating potential oversights and mistakes and in making the reviewer's job easier.

In the future we are looking forward to steps for systematic incorporation of design-in-depth features.

REFERENCES.

1. Rebecca Menes and Herb Hecht, "Safety and Certification of UAVs", SAE Aerotech Symposium 2007.
2. Theresa Robinson and Nayden Kambouchev, MATLAB/Simulink Tutorial, web.mit.edu/16.060/www/handouts/Matlabtut.pdf, 2004
3. Federal Aviation Administration, Advisory Circular AC 25.1309-1A "System Design and Analysis", 6/21/88. This circular has now been adopted by international bodies.
4. Idaho National Laboratory, Control Systems Cyber \security: Defense in Depth Strategies, Department of Homeland Security, May 2006
5. Tony Savor and Rudolph E. Saviora, "Toward Automatic Detection of Computer Failures", IEEE Computer, August 1998, pp. 68-74
6. Jana Mulacova, Failure Detection Expert Software, Diploma Thesis, Czech Technical University, Prague, 2007
7. SAE International, Guidelines and Methods for conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, ARP4761, December 1996
8. Herbert Hecht and Patrick Crane, Rare Conditions and their Effect on Software Failures, Proc. of the 1994 Reliability and Maintainability Symposium, pp.334-337, January 1994.
9. C. K. Hansen, *The Status of Reliability Engineering Technology 2001*, Newsletter of the IEEE Reliability Society

CONTACT

Herb Hecht, SoHaR Inc. Tel: 310-338-0990
www.sohar.com
5731 W. Slauson Ave. Suite 175, Culver City, CA 90230
herb@sohar.com
Rebecca Menes, Address as above, becky@sohar.com