ESMoL Fault Modeling Use Cases and Principles

Last Modified: 10/11/2010 – Joe P.

TODO:

1. Come up with term algebraic descriptions for behavioral models behind ESMoL, and characterize the variance due to uncertainties. Maybe use an abstract behavior parameter to represent the behavioral effects of another related domain on the domain we're interested in analyzing. Find behavioral congruences under the uncertain models, and map those to ESMoL models. What do we find out?

Infrastructure Use Cases for Fault Modeling

- Relate fault models to captured requirements
- Generate a Simulink simulation of any fault that we can model
- Generate code for fault injection on the controller side
- Monitor generation? - we would have to extend the modeling language to associate functional blocks for monitoring/mitigation with faults and requirements
- Test case generation – input data, expected output data, correctness conditions
    - Regression tests for bugs – associate with bug reports, we would also need some way to represent model versions and their associated generated software to correlate the bugs to something meaningful
    - Scripts for running tests, comparing outputs with expected results, evaluating correctness conditions, and collecting statistical results.

Concept Notes

- Catastrophic failure – can not get to any safe state in any number of hyperperiods
- Recoverable failure – get to "walk" (vs. "run") mode to buy recovery time (e.g. hover)
    - Number of hyperperiods required/allowed to get to safe mode and then to full operation or landing
    - Scheduling questions in recovery mode
        - What is the recovery function utilization? Can we still run controller functions?
        - What about multiple failures? What cases do we need to cover to guarantee schedulability?
- Schedulability analysis

    - We are not using the usual approach, where we leave permanent slack for fault handlers
        - Fault detectors must run in-band with the controller functions
        - We assume that fault mitigation is a different mode, and do the schedulability analysis separately from the nominal/low-priority on-demand function to get an idea of what's possible in recovery mode
- Detection functions: thresholds, differential thresholds (rate limits), ...

Fault Modeling

Elements:

Requirements

Events/Conditions

Scenarios – fault probabilities, inputs and outputs to exercise conditions

Relevant model elements: detectors, handlers, signals (for fault injection), hardware and software components (and their failure modes)

Tests – input representation, probabilistic simulation of the fault, output/data capture, analysis

Fault cases and ideas:

Scheduling faults

Fault	**Detection**	**Mitigation**
Failure to complete task	FRODO expectation check Output messages not delivered	Notify and try again (next HP) If restart, recover and land If no restart, then based on which task failed we may crash soon We'll need to model task criticality and point of no return time
WCET overrun	FRODO expectation check	Notify May not be a problem -- we should have a slack parameter from the scheduling analysis which indicates whether we have impacted another component. within slack -> ignore outside slack -> recoverable failure repeatedly outside slack -> may not be recoverable
Deadline miss	FRODO expectation check	We may want to consolidate this case with the last one. The detection should be on the deadline.
Loss of time sync	FRODO expectation check	Notify everyone and try to land Bad things should happen quickly in this case.

Communication Faults

Fault	**Detection**	**Mitigation**
Failure to consume new data		Try to restart local tasks?
Message sent at the wrong time	FRODO	Notify and try again (next HP) Has someone lost track of the time sync? How would we find out?
Wrong message ID sent	FRODO	Notify. Is there a way to figure out if someone else has lost time sync? May need to reset everyone and re-sync. If unsuccessful, crash.
Bad data on arrival	Need to add a simple crc16 or something similar	Notify and try again (next HP). Keep count of successive bad data packets. If too many, try to land.
Failure to receive message expected at particular time	FRODO	Notify and try again (next HP) Check comm link?

Hardware Faults

Fault	**Detection**	**Mitigation**
Processor dies	Other nodes no longer receive messages	Try a hard reset? Crash? Can the other nodes land it?
Power out	Hardware support (sensor?)	Crash
Loss of connectivity	FRODO	Can we land still, or at least get to hover mode?

Functional Faults

Fault	**Detection**	**Mitigation**
Bad input values	Precondition bound checks	Notify and try again (next HP) Reuse the old input value once or twice, and then fail. Try to land.
Bad calculation	Post condition bound checks	same as above, but reuse output value

Bad result		Is this basically the same as bad calculation?
Bad state	?	Can we reset components and/or subsystems to get into a known good state?

Scenario Modeling

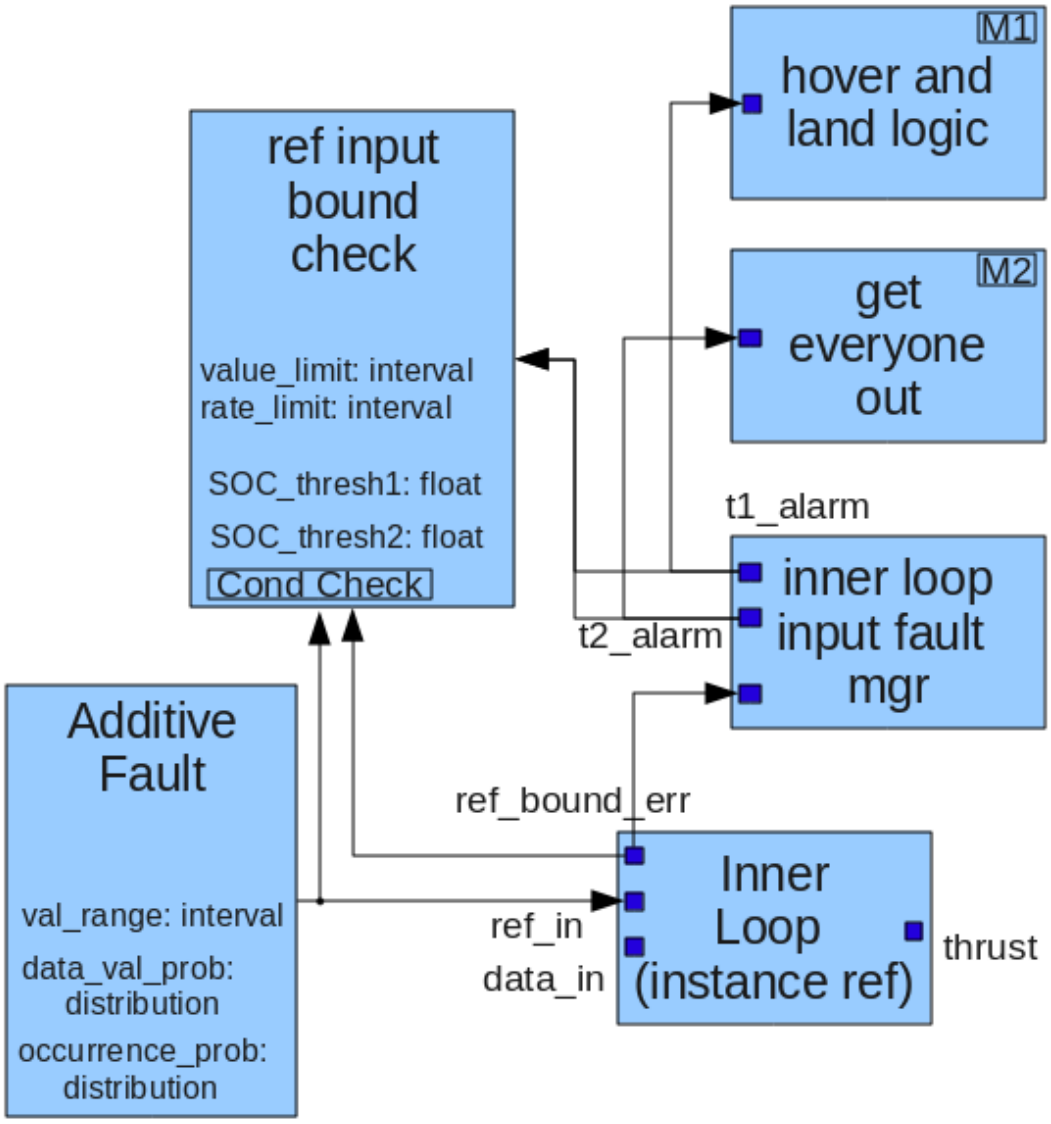
Working on representing all of the necessary information to describe a particular fault and cover the toolchain use cases.

Scenario 5.3: Bad reference input for inner loop

- Criticality: A or B (in the sense of DO-178B, where A is worst)
- Requirements: 127.3.2, 971.5.4 (dummy examples -- use strings)
- Events:
  - inner loop function triggered to execute
- Scenario Preconditions:
  - incoming data was delivered on time
  - incoming data has passed checksum
- Detection:
  - value out of bounds of specified valid range for precondition
    - absolute out of bounds (e.g. 3 is not in the interval [0,2])
    - differential out of bounds (e.g. new value 1.87 exceeds the rate limit when compared to the previous value 0.5)
  - detector actions
    - notify FRODO
    - increment successive occurrence count (SOC) (reset on every good value)
- Mitigation:
  - if SOC < threshold1
    - reuse previous value
  - if SOC >= threshold1
    - try to hover, then land
  - if SOC >= threshold2
    - bail out, we're fixin' to crash
- Test cases

- Hovering
- Climbing
- Banking turns
- Landing

Possible informal graphical representation of fault model:



Here the precondition check and notification logic is in the inner loop, but the fault collection and counting is outside in a different task. The relationship between controller tasks and fault managers may be N:1 or N:M.

We could use a similar notation to define test cases, where different blocks represent the input and output functions that we want. Merging fault models and test cases should then be straightforward to create simulations.

From Project Discussion on 9/13/2010:

FMECA – spreadsheet-driven fault modeling

Enumerate fault scenarios, effects, and criticality

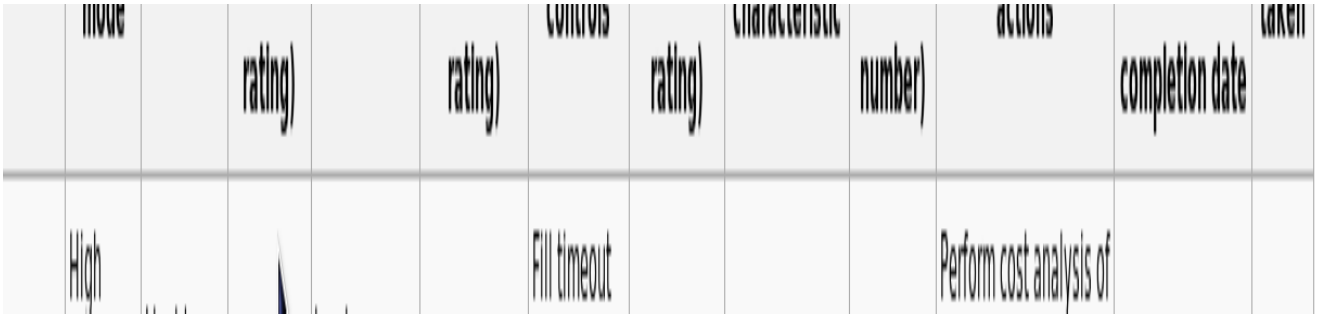
Use the table to drive analysis and testing

From Mil Reliability Guide:

No.	Unit	Function	Failure Mode	Probable Cause	Effect On			Interrupt ?	Crit	Action
					Unit	Sub	System			
1	Output	Outputs file into	Output is incorrect	Inputs are invalid and not detected	n/a	none	mission degraded	no	II	lab repair
2	Output	Outputs file into	Output is incorrect	Inputs are correct but not stored properly	n/a	none	mission degraded	no	II	lab repair
3	Output	Outputs file into	Output is incorrect	Values are not computed to spec	n/a	none	mission degraded	no	II	lab repair

FIGURE 9.8-1: EXAMPLE OF SOFTWARE FMECA

From Wikipedia (FMEA)



Wikipedia example: FMEA, with risk priority numbers -- 1-10 each for severity, occurrence, detection

RPN = S x O x D

Help prioritize the faults and direct development efforts.

Questions / Discussion

- Is the spreadsheet a good modeling format? It helps with systematic coverage and categorization of all faults we can think of.
- Modeling use cases – what questions must be answered by the models?
  - To what level have we detailed the causes/effects of each fault?
  - To what degree have we mitigated each fault?
  - Do we know fault severity, occurrence, and detectability?
  - How does the FMECA worksheet information correlate with what we have in the ESMoL model? What information do we need to add to ESMoL?
- Use cases – can we use statistical model checking with fault scenarios to analyze confidence for the design?
- What about test coverage?
- What information needs to go into test descriptions?
- Modeling: Is it simpler/useful to simply refer to external test specs?

Discussion from Meeting

Gabor:

Severity can only be determined by relating fault triggers to their effects. Occurrence is basically the failure rate of a particular component or system. The problem with detectability is that many faults are not directly observable. The question is whether and how we can detect the failure mode from the effects?

In analyzing the system, we propagate the fault effects forward, and we propagate critical failure conditions backwards. We want to make sure that no fault effects can reach any critical failure conditions. Look at Nancy Leveson – Safeweb.

Fault (cause) ---> Anomaly (detection) ----> Failure (effect)

Mitigation strategies are often undocumented, which is why the exhaustive FMECA-style evaluation is important. We should visit every failure case.

Testing – hard problem. Exhaustive testing is impossible. Even directed testing gives too many cases to actually run. We want to spend our limited testing time on parts that are absolutely critical. The hard questions are

1. How do we close the loop? (need to clarify this question) and 2. How do we mitigate faults (and be sure about it)?

From IEEE Control Systems Magazine, Oct. 2010:

James Spall. Factorial Design for Efficient Experimentation. (p. 38-53). This describes on a model-focused (i.e. parametric models) approach to test design, where we start with a basic model structure and a nominal design. Then we want to change factors between tests in order to discover the main effects of the changes as well as the interactions between inputs. He covers the basics of factorial design, where the full range of combinations of possible simultaneous input (or parameter) changes between tests is covered. Compared with one-factor-at-a-time experimentation, the information per test scales phenomenally better in factorial test designs. The ratio of required tests for non-factorial methods to factorial methods is linear with the number of factors.

For our applications there are some good ideas here, but they need refinement (i.e., long-term efforts).

1. For example, could we create a set of parametric model templates describing the behaviors of our building blocks? We could automatically and symbolically assemble a large parametric model, but then we need to prune it in a smart way.
2. What about interface guarantees? Can we use interface profiles to store information about effects and interactions in order to reduce the integration testing burdens?

Ideas

One of the most common important operations is that of comparing behaviors. This may be done via collected traces of either different simulation models, or via traces from a running system. To automate this process we need conditions to check against, and the ability to script simulations or actual hardware tests. Repeatability is important to assess for any particular test case.

Gabor/T-Vec – CSP to create test vectors

Walters/Suri – EDICT fault modeling and analysis environment

Take a look at Christopher Price, Effortless Incremental Design FMEA. (IEEE Xplore) He describes a modeling tool (FLAME) that includes FMEA descriptions which are tied to parts of the circuit. When the circuit design changes, the topology changes are isolated and then the user is presented with the elements of the FMEA that have changed. Also Throop, D.R. ; authorLink(" Malin, J.T.");Malin, J.T. ; authorLink(" Fleming, L.D.");Fleming, L.D. ; , Automated Incremental Design FMEA.

Also look at Incremental fault-tolerant design in an object-oriented setting. E.B. Johnsen, O. Owe, E. Munth-Kaas, and J. Vain. Quality Software 2001. p. 223-230. This approach requires formalized requirements, but operates on communication traces and proceeds to a fault-tolerant design by refining a non-fault-tolerant design.

For refinement, maybe consider authorLink("Cailotto, S.");Cailotto, S. ; authorLink(" Fin, A.");Fin,

A. ; authorLink(" Fummi, F.");Fummi, F. ; A fault tolerant incremental design methodology.