IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727 Volume 5, Issue 6 (Sep-Oct. 2012), PP 15-20
www.iosrjournals.org

Applying a Model Based Testing for a Controller Design in Fault Detection, Isolation & Recovery (FDIR) System

Venkatesh Kumar. N[1], Amritha Mulay. V[2], Mujeeb Ulla Jeelani [3]
[1] *Asst. Professor, SET, JAIN University, Bangalore, Freelance consultant and Trainer.*
[2] *Software Engineer, Allied Tools and Electronics, Bangalore.*
[3]*Software Engineer, 7Star Technologies, Bangalore.*

Abstract: *Model-Based Design allows the process of verification and validation of an executable system specification, which prevents errors from persisting into later stages of the design process where they are more costly and time consuming to fix. Studying a Fault Detection, Isolation, and Recovery (FDIR) system of a controller design, in this case an aircraft elevator redundancy control system is considered, which demonstrates how to trace requirements to a design, use the test cases based on those requirements, and perform the coverage analysis, which in turn reveals untested, missing, and ambiguous requirements in the specification.*
Key words: *Model Based Design, Graphical Model, Actuator, hydraulics.*

I. Introduction

With Model-Based Design, a graphical model, often a block diagram is used to capture the system requirements. This model produces an executable specification that describes the system behavior and can be gradually extended into an increasingly detailed design [1] [2]. Verification and validation can occur earlier in the system design process, which reduces costly iterations across many design steps.

The creation of consistent, reusable, and well-documented models becomes the important stage in the development of embedded systems. Hence, the concept of model-based development (MBD) emerges. Due to the increasing complexity of the developed systems it is necessary to model correctly and to implement the chosen design in a correct manner.

In general, the requirements for engineered systems are ambiguous, not rigorous, and even inconsistent. It is desirable to detect such problems as early as possible in the system design process. Model-Based approach can be used to achieve this goal by facilitating executable specifications that allow testing to start at the model level instead of after system realization.

II. Description of an Elevator Control System

A typical aircraft has two elevators attached on the horizontal tails (one on each side of the fuselage). There are a number of redundant parts in the system to ensure safety [2].
For example, as shown in Figure 1, there are:

- Two independent hydraulic actuators per elevator (four in total).
- Three separate hydraulic circuits to drive the actuators.
- Two primary flight control units (PFCU).
- Two control modules per actuator: Full range control law and limited / reduced range control law.

In this system we limit each PFCU to have only one control law, and to control only one set of actuators to reduce the complexity of the system. A PFCU having a sophisticated Input-Output (I/O) control law, controls the left (LIO) and right (RIO) outer actuator. In case of a failure, a Direct-Link (DL) control law with reduced functionality handles the left (LDL) and right (RDL) inner actuators. Each outer actuator has a dedicated hydraulic circuit, whereas the inner actuators share one hydraulic circuit. By default, the outer actuators are on, and the inner actuators are on standby. If a fault is detected in the outer actuators or in the hydraulic circuits that are connected to them, we want the system to respond accordingly to maintain stability by turning the outer actuators off and activating the inner actuators [2]. The mode logic block operates the redundancy to assure continual operation of the system.

This paper focuses on the fault detection, isolation, and recovery logic that causes the actuators to switch from one mode to another. Here simple yet representative models of the hydraulic actuators and elevators, as well as the feedback control laws are presented, to illustrate the behavior of this logic.

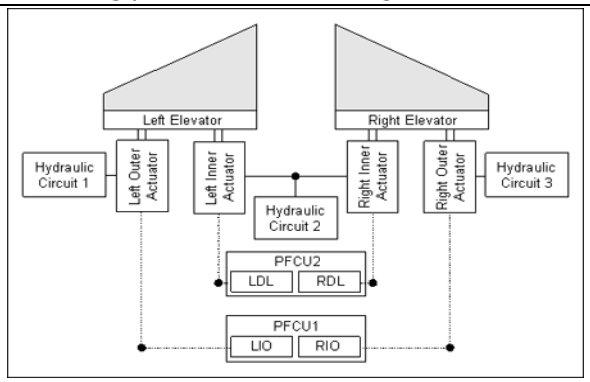

Figure 1 Components of the elevator redundancy system

III. System requirements

As mentioned, there are four actuators in total, two on each elevator. Each actuator has five modes:

- **Isolated mode.** The actuator is turned off indefinitely. An actuator must be able to be isolated from any mode.
- **Off mode.** The actuator is turned off temporarily because of a failure and will come back online only if all failures for that actuator are fixed.
- **Passive mode.** The actuator is waiting and does not generate actuator control signals. An actuator in passive mode can take control of the elevator control system.
- **Standby mode.** The actuator's control law is active, but no force is applied to the actuator. It is ready to take control of the system and will introduce minimal transients when it does.
- **Active mode.** The actuator is active and controls the elevator.

IV. Control logic system structure

We know that the system contains four actuators, as shown in Figure 2, each of which can be in one of five modes: active, standby, off, passive, or isolated. To implement the mode logic in Stateflow, we create four parallel superstates that represent the actuators, and within each superstate are five exclusive states that represent the five modes given above. The actuator superstates are parallel because the actuators run concurrently in the system [4]. On the other hand, the five mode states are exclusive because an actuator is in one and only one mode at any given time.

The operator would not want this actuator to be active right away because;

1) To run the risk of using two actuators simultaneously to control one elevator,
2) The operator is trying to minimize mode switching, and
3) The operator would like the elevators to be symmetric if possible (i.e., either both inner actuators are active or both outer actuators are active).

So if the inner actuators are active, they should remain active even if one of the outer actuators has come back online. A better alternative is to set the actuator to the passive mode by default.

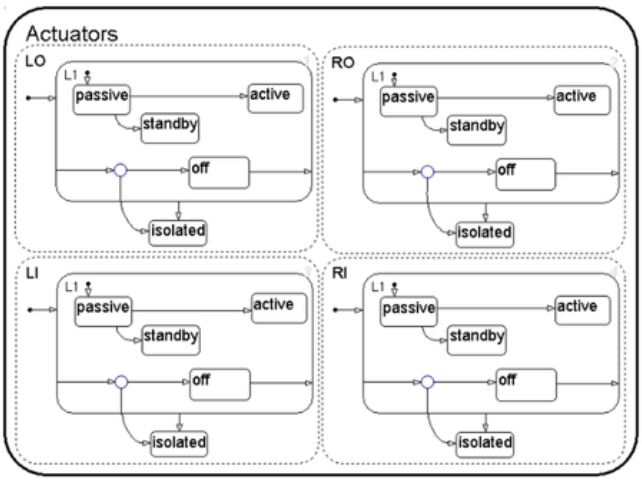

Figure 2 Complete actuator control logic system without FDIR capability.

V. Fault detection and isolation

Table 1 shows a small set of high-level requirements used to guide the design of the mode-logic component. Such high-level requirements can be incomplete, inconsistent, and difficult to interpret. Design errors can be introduced simply by misinterpreting the high-level requirements into more detailed requirements [6]. For example, the combination of requirements 2 and 3 from Table 1 creates an ambiguity, i.e., If one actuator can be operated only in inner loop, should the other still be operated in outer loop.

Therefore, it is necessary to test the detailed requirements early in the development process to ensure their accuracy. Hence to design an implementation, the high level system requirements needs to be first translated into more detailed, subsystem level, requirements.

Part of the detailed requirements that can then be translated into (formal) specifications is shown in Table 2. Based on these specifications, a controller model is designed that embodies the desired behavior. The design then needs to be tested against those requirements to verify and validate its compliance [6]. The approach demonstrated revolves around being able to independently establish test cases that are derived from the high and low-level requirements that can be executed on the design model, that has been created based on the same requirements, to verify that the requirements have been met. The ability to have full traceability of the high and low-level requirements to the design model through software implementation is achieved. Establishing such a formalized approach to testing the design is important to demonstrate that it meets the written requirements.

ID	Description
1	Each actuator will have five modes: Isolated, Off, Passive, Standby, and Active.
2	If possible, the same control law should be active for both the left and right elevators.
3	If available, the I/O control law should be active instead of the DL control law.
4	The actuator that is not active should be in standby.
5	If the pressure of the hydraulic circuit is low and the position measurement fails, the corresponding actuator should be switched to Off.
6	If the pressure of the hydraulic circuit is nominal and the position measurement fails, the corresponding actuator should be switched to Isolated.
7	Controller state changes should be made only in response to failure events.

Table 1 Small set of high-level requirements

VI. Fault definition

We have listed the requirements that this system must fulfil with respect to fault detection. If the aircraft is flying perfectly level, then the actuator position should maintain a constant value. If the position of an actuator increases or decreases by 10 cm from zero point, then a failure has occurred in that actuator. A failure also occurs if the change in position is very rapid. Similarly, a fault occurs in one of the hydraulic circuits if the pressure is out-of-bounds or if the pressure changes very rapidly.

The fault detection subsystem that is implemented in Simulink is as shown in Figure 3. The actuator position and hydraulic pressure values are input into both the Rate of Change Threshold and Range Threshold blocks, which then determine whether or not these values are valid [6]. This information then gets sent to the actuator mode logic system. If a failure has been detected, then the mode logic system will react accordingly (e.g., an actuator can be switched off).

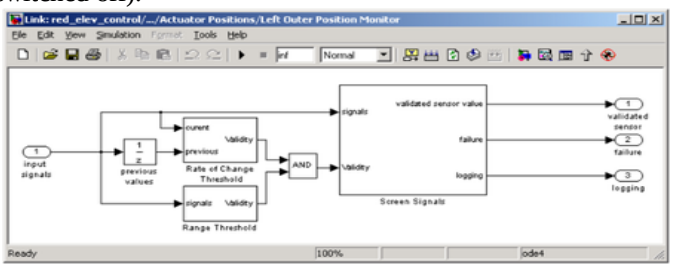

Figure 3 Fault detection system

VII. Using Truth Tables for fault detection

As we can see above, the logic system needs to fulfil a significant number of requirements. One way to incorporate each of these requirements into Stateflow is to use truth tables. Truth tables are used to realize logical decision-making behaviour. Stateflow truth tables contain conditions, decisions, and actions, as shown in Table 3.

Let's look at the first requirement we listed in this part to see how it fits into the truth table. To repeat, the first requirement is that if a failure is detected in the hydraulic pressure 1 system, while there are no other failures, we turn off the left outer actuator. This requirement is met through Decision 1 of the condition table. Note that in Decision 1, the second condition (left outer actuator failed) is blank, meaning that the condition can be either true or false.

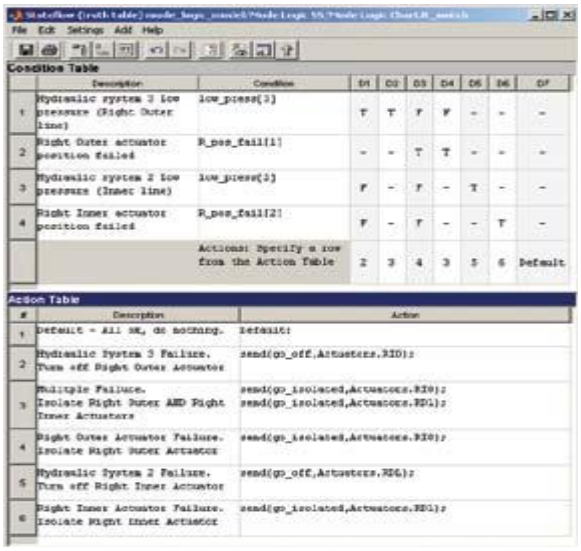

Table 3 Truth table for left elevator

ID	CONDITION	DESCRIPTION
1	Hydraulic pressure 1 failure	If a failure is detected in the hydraulic pressure 1 system, while there are no other failures, isolate the fault by switching the Left Outer actuator to the off mode.
2	Hydraulic pressure 1 fails and then recovers	If a failure is detected in the hydraulic pressure 1 system and the system then recovers, switch the Left Outer actuator to the standby mode.
3	Hydraulic pressure 2 failure	If a failure is detected in the hydraulic pressure 2 system, while there are no other failures, isolate the fault by switching the Left Inner actuator and the Right Inner actuator to the off mode.
4	Hydraulic pressure 2 fails and then recovers	If a failure is detected in the hydraulic pressure 2 system and the system then recovers, switch the Left Inner actuator and the Right Inner actuator to the standby mode.
5	Hydraulic pressure 3 failure	If a failure is detected in the hydraulic pressure 3 system, while there are no other failures, isolate the fault by switching the Right Outer actuator to the off mode.
6	Hydraulic pressure 3 fails and then recovers	If a failure is detected in the hydraulic pressure 3 system and the system then recovers, switch the Right Outer actuator to the standby mode.
7	Default start-up condition	If there have been no failures detected, the Outer actuators have priority over the Inner actuators. Therefore the elevator actuators should default to the following modes. The Left Outer and Right Outer actuators should transition from the Passive mode to the active mode. The Left Inner and Right Inner actuators should transition from the Passive mode to the Standby mode.

Table 2 List of the detailed requirements for the mode logic design

VIII. Recovery

This part of models in an aircraft elevator control system using Model-Based Design with Simulink Stateflow and Simulink Verification and Validation is accomplished. We design an actuator controller from a set of requirements and then verify that the controller meets these requirements through appropriate test cases. We can apply the design approach discussed in this case study to other event-based logic systems.

We designed the overall structure of the elevator control system. In Previous section, we modified the control system to allow for the detection of faults in the actuators or hydraulic circuits and the isolation of specific actuators based on these faults. In this section, we modify the logic system once again so that actuator subsystems can be recovered once specific faults are no longer detected.

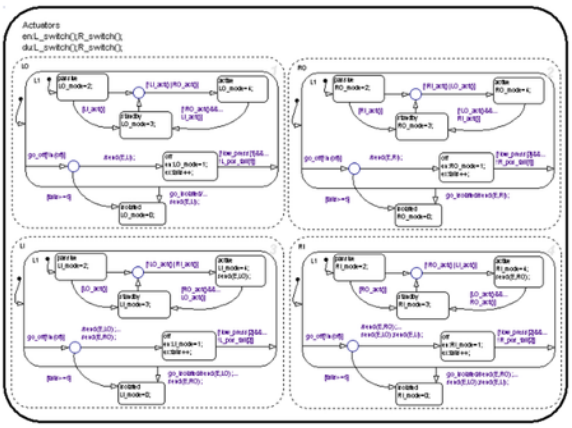

Figure 4 Complete FDIR actuator logic system

Now that we have the recovery requirements for the actuator control system, we can modify the Stateflow chart accordingly. The first step is to add transitions to the chart that are necessary for recovery purposes. We will add transitions to satisfy each recovery requirement by adding a transition label appropriate for each requirement.

It will be necessary to transition from Active to Standby mode and vice versa. For example, look at what happens when the left outer actuator is switched off due to a failure. In this case, the right outer actuator is switched from Active to Standby mode, and the inner actuators are switched from Standby to Active mode. Tests designed for all requirements can be combined into a test harness and applied to the design model for requirements-based testing. Tests can be executed individually or in batch mode. If a test runs without any verification blocks asserting, then the design passes the test. If an assertion is detected, the simulation stops and the verification block issuing the assertion is highlighted, which helps diagnose why the test failed.

IX. Coverage Analysis

Creating and executing requirements-based tests to ensure that the design behaves as expected is not the same as fully testing the design. Some requirements may lack tests, the requirements themselves may be ambiguous or incomplete, and the design may contain superfluous elements.

By using Simulink Verification and Validation, coverage metrics can be collected during simulation to indicate untested design elements. The coverage metrics are displayed directly in the model using coloured highlights and in a dialog box with summary information. Additionally, a detailed report of the coverage analysis is created. The coverage metrics collected include cyclomatic complexity, decision coverage, condition coverage, MC/DC, lookup table coverage, and signal range coverage.

X. Requirement based Testing

Let's look at the following specific requirement in order to understand how requirements are tested:
If a failure is detected in both the hydraulic pressure 1 system and the left outer actuator, while there are no other failures, turn off the left outer actuator. First we generate a test case for this requirement in a Signal Builder block within the Test Cases block, as shown in Figure 5.

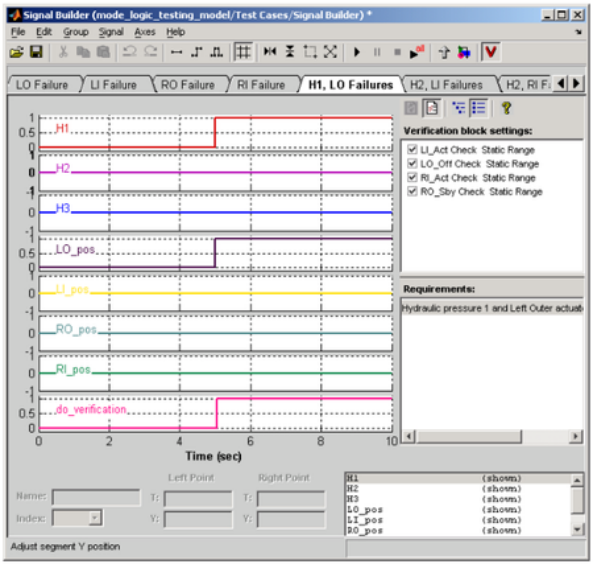

Figure 5 Test case in Signal Builder Block

There are a total of eight signals shown in Figure 5 three signals for each hydraulic circuit, four signals for each actuator, and one signal to switch on the verification process. Let's look at the first seven signals. The test cases have been set up so that a value of 1 corresponds to a failure, and a value of 0 corresponds to a nominal value. For this requirement, there are failures in both hydraulic circuit 1 and the left outer actuator. Therefore, you see both of those signals start at 0 and then change to 1 as a failure is detected. The values of the other signals for the hydraulic circuits and actuators stay constant at 0 since no other failures are detected.

For the last signal labelled do_verification, when the value is 1, we check to see whether the actual modes of the actuators correspond to the theoretical modes of the actuators. The expected modes for each actuator are specified in the section labelled Verification block settings in the Signal Builder block (upper-right section in Figure 5). In this, we expect the inner actuators to be active, the left outer actuator to be turned off, and the right outer actuator to be on standby.

XI. Results and conclusion

System design often starts off with a set of high-level requirements (e.g., 'mission requirements') that are gradually refined into a set of detailed requirements from which the subsystem specifications can be derived. Once the subsystem specifications are available, an implementation can be designed. Once the implementation has been realized, extensive testing determines whether the original requirements are met.

In general, requirements are ambiguous, not very rigorous, and even inconsistent. It is desirable to find the problems as early on in the design process as possible. Model-Based Design can be used to that goal by facilitating executable specifications that allow the testing otherwise done after system realization.

References

[1] Jason Ghidella and Pieter J. Mosterman Requirements-Based Testing in Aircraft Control Design, The MathWorks, Inc., Natick, MA, 01760, USA.
[2] Pieter J. Mosterman, Manuel A. Pereira Remelhe, Sebastian Engell, and Martin Otter, Simulation for Analysis of Aircraft Elevator Feedback and Redundancy Control.
[3] Simulink, Model-Based and System-Based Design © COPYRIGHT 1990 - 2004 by The MathWorks, Inc.
[4] Stateflow® Getting Started Guide ©COPYRIGHT 2004–2012 by The MathWorks, Inc.
[5] Politecnico Di Milano - Dipartimento Di Ingegneria Aerospaziale Aircraft Systems –Chapter 6 – Flight Control System LECTURE NOTES, VERSION 2004.
[6] Dan Ye and Guang-Hong Yang, Senior Member, IEEE, Adaptive Fault-Tolerant Tracking Control Against Actuator Faults With Application to Flight Control. IEEE Transactions On Control Systems Technology, VOL. 14, NO. 6, NOVEMBER 2006.
[7] Bo Yang, Fei Zhao, Xiayun Zhao, Sen Yang Simulation Based Design of a High-Speed +Elevator System. Spring 2008.