# Study of Fault Scenarios in Aerospace through Model Reuse

Dr. Jason Ghidella, Dr. Pieter J. Mosterman
*The MathWorks, Inc.*
*pieter.mosterman@mathworks.com*
*jason.ghidella@mathworks.com*

Bradley Horton
*The MathWorks Australia Pty Ltd*
*bradley.horton@mathworks.com.au*

**Abstract.** Throughout the aircraft design process, engineers expend considerable effort modelling every aspect of the vehicle and its subsystems. Rather than creating single-purpose models however, designers could extend much of this work to improve the design quality and productivity in other areas. When these models are designed with custom-off-the-shelf (COTS) tools, they can be reused elsewhere in the design process. Real-time code is instrumental in this effort as it provides the means to integrate the high level control laws and system models with other existing software. This real-time code can be generated automatically from the same models used in the aircraft design stages, and then reused again for different studies such as hardware-in-the-loop simulation, software-in-the-loop simulation, and rapid control prototyping. This paper discusses model reuse and shows, as an example, how discrete-event models of the reconfiguration management system of an aircraft elevator control system, can be connected to the dynamic hydraulic and mechanical models of the elevators and their different feedback control laws. Combining the supervisory control logic with the continuous plant behaviour allows simulation of the transients in elevator deflection when a hydraulic line failure occurs. Although these transients are minimal if a redundant actuator is able to quickly respond from a standby mode, significant transients including instabilities, could be introduced if the redundant actuator is in a passive or off mode. Through automatic code generation these same models can then be reused and deployed onto aircraft training simulators, where aircrew can gain familiarity with the performance of the aircraft during a system failure, as well as learn the appropriate isolation and recovery procedures handling the failure.

## 1. INTRODUCTION

In aircraft design and development, the use of Model-Based Design has become widespread. Models are being used to design, analyse, optimize, and synthesize hardware and software systems. For example, in terms of hardware, a model of a hydraulic actuator may be used to verify that a particular design, its dimensions, connections, and configuration, satisfies the requirements such as delivered power and response. In terms of software, a model of a feedback control law may be used to verify that a particular choice of sample rate and fixed-point resolution satisfies requirements, such as stability and robustness. Though widespread, the use of models may still be very fragmented in an organization. Different teams of engineers are involved in the different stages of aircraft design and each team may use models that they have designed themselves. In many cases, this is because there is no one common tool available that allows transformation, or even correlation, of models that capture different system perspectives. For example, at present, there is modest support to relate a SolidWorks® [1] Computer Aided Design (CAD) model to a Simulink® [2] model. Though a SolidWorks model can be imported in SimMechanics [3] it cannot be exported back for further editing.

On other occasions, however, the same formalism or even tool may be used by different design teams, with great potential benefit for model reuse. Partially, this may require enculturation, as it is tempting to design a model for one's own purposes without trying to produce a more general representation of the system it aims to capture. It is not uncommon to design a model of a physical system based on a particular scenario in which it is used. For example, a number of fixed sequences of valve openings and closings may be built into the model of a hydraulic actuator, and so an extraordinary situation that was not pre-conceived, and that does not occur during nominal operation, is not included. As such, the model can be used in the design of, say, the reactive supervisory control, but it would be of no use for reconfiguration control in the face of failures.

In addition to removing as many built-in assumptions as is economical, the tool used to design the models and the formalism used for the model representation determine the re-usability of this model. Though many industrial design efforts have standardized on MATLAB®[4] and Simulink as their modelling and simulation environment, efforts are under way to allow model exchange between tools and the transformation of models into different formalisms [5]. A more straightforward yet versatile approach to combine and integrate models is by exploiting the code generation facilities that are available in many modelling tools. By transforming the different models into C/C++ code, they can be compiled into one executable, which requires nothing more than some 'glue code' [14]. This approach

is especially viable in cases where real-time execution is a critical requirement.

This paper discusses the reuse of a model created for the design of a reactive controller in Stateflow® [10] to manage the switching of redundant actuators. In particular, the actuators are used to position the control surfaces of the HL-20 crew rescue vehicle, which was developed at NASA Langley [8]. Once a model of the reactive control has been designed, the switching between modes is analysed, studying the transient effects. This analysis requires detailed continuous-time models of the dynamics of the hydraulic actuators and the mechanics of the elevator system that they are attached to. The same controller model is then directly integrated into a model of the HL-20. This model connects to a FlightGear [9] simulation and provides a visualization of flight behaviour. Similarly, real-time code can be generated from the controller model for different targets and integrated with real-time flight simulators.

Section 2 of this paper first presents the redundancy control of the elevator control system. In Section 3 the dynamics of the hydraulic system, mechanical linkage, and feedback control are included in the system model to study transient behaviour in the face of faults. In Section 4 the system model is further extended by including interacting systems present in the HL-20. Section 5 discusses the generation of real-time capable code. In Section 6 the conclusions are given.

## 2. REDUNDANCY MANAGEMENT

*Figure 1 shows a simplified configuration of the redundancy present in the elevator system of civil aircraft. It consists of two elevators, one on either side of the vehicle. Each elevator can be positioned by two actuators, only one of which should be active at any given time. The four actuators are driven by three separate hydraulic circuits, as shown in *Figure 1.

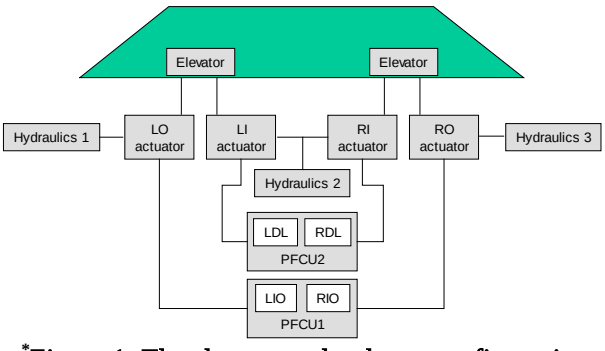


*Figure 1: The elevator redundancy configuration

Two primary flight control units (PFCU) control either the inner or outer actuators, each of which employ a separate control law. In nominal operation, a sophisticated input-output (IO) control law is applied to the left (LO) and the right (RO) outer actuators. In case of a failure, a direct link (DL) control law with reduced functionality is available for the left (LI) and right (RI) inner actuators. A more realistic and extensive version of this system has been presented in other work [6],[7]. *Figure 2 shows a Stateflow chart that implements the mode logic for the left actuators. Here the operating modes of the outer actuator are described by the LO parent state, and the operating modes of the inner actuator are described by the LI parent state. The structure of the right actuators mode logic (RO and RI) is identical; however, the transition conditions and the variables they set are different.

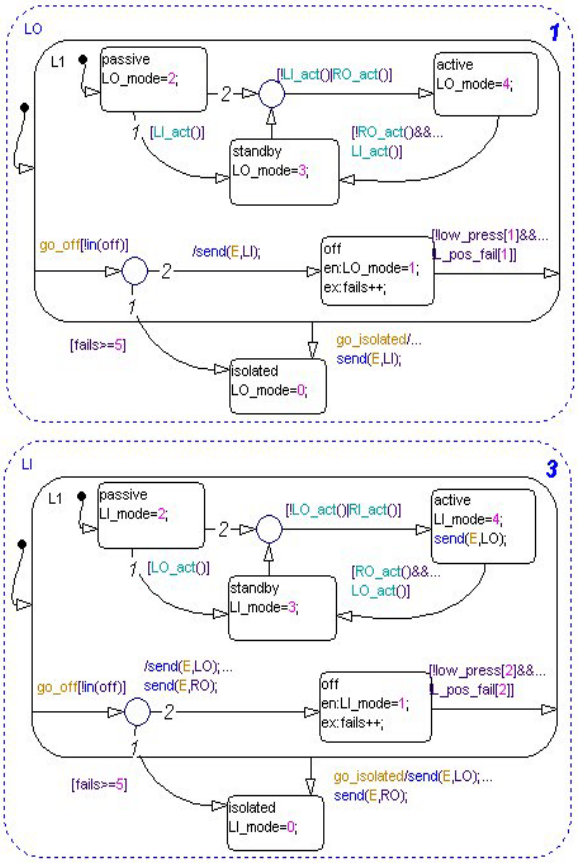


*Figure 2: Left actuators mode switching logic

The execution order of the four parallel state transition diagrams has been deterministically set so that at each evaluation of the statechart, the LO module is executed first, then the RO module, then the LI module, and finally the RI module. This order of execution is critical in determining the logic for the recovery aspects of the system. When the PFCUs are powered up, the state machine takes the entry transitions (commonly referred to as default transitions, and represented in the diagram as arrows with a solid circle on one end) that move them into the *passive* state for each actuator. This can be seen in *Figure 2, which also shows that when this occurs, the variables LO_mode and LI_mode are set to the value 2. Analogously, the other two parallel states set the variables RO_mode, and RI_mode to 2 as well. From this *passive* state, the four modules infer their active states.

The LO module moves to *active* if the LI module is not in *active* or if the RO module is *active* (represented by the condition `[!LI_act()|RO_act()]`), otherwise the LO module will move into *standby*. The first condition (`!LI_act()`) is to ensure there is always one active

actuator for each elevator. The second condition (`RO_act()`) is to ensure symmetry, that is, if the right elevator is operated by the IO module, then the left elevator should also be operated by its IO module, if possible. These switching conditions are similar for the other three actuators. Now, because of the deterministic execution order of the Stateflow chart, if there are no failures in the system, the LO and RO modules will move to *active*, while the LI and RI modules will transition into *standby*.

An important aspect of the actuator mode logic model is the use of hierarchy. For example, in the LO module shown in *Figure 2, hierarchy allows entering *isolated* from all other states, and the LO module can go *off* from *passive*, *standby*, or *active*. Furthermore, as the LO module transitions to either *off* or *isolated*, events are broadcast directly to the other actuator modules to ensure that they are in the correct mode. This is illustrated by the transition to *isolated* in the LO module. This transition broadcasts the event `E` to the LI module to ensure that it will take over control of the left elevator and that the system will be in the correct configuration when the remaining parallel states have executed for the given time step. The module can recover from *off*, given the right condition (no failures in the hydraulic pressure and actuator position), but the module cannot recover from *isolated*. Consider the scenario where the LO module moves into *isolated*. During this transition, event `E` is broadcast to the LI module, causing it to evaluate the transition guarded by the [`!LO_act()|RI_act()`] condition. This moves the LI module from *standby* to *active*. The LO module then completes its transition into *isolated*. The next evaluation in the state chart that occurs is in the RO module, currently in *active*. Given that its transition from this state is guarded by the condition [`!LO act()&&RI_act()`] is not satisfied, it stays in *active* for now. The LI module then evaluates a similar condition, [`RO act()&&LO_act()`], as false, and therefore, LI remains in *active*. The RI module, however, takes the transition guarded by the [`!RO act()|LI_act()`] condition and moves into *active* since the LI module is in *active*. As the RI module enters *active*, it broadcasts an event E to the RO module causing it to take the transition guarded by the condition [`!LO_act()&&RI_act()`], (that now evaluates to true), out of *active* into *standby*.

## 3. SWITCHING TRANSIENTS

Once the redundancy management has been formalized in the controller model, the mode switching effects on transient behaviour must be studied. To this end, the mode logic model must be connected to a closed loop plant model that captures the behaviour of the physical world, in this case, the hydraulics and mechanics of the elevator system, as well as the feedback control system.

Different teams of engineers are usually responsible for the plant and control system design. In the case of our example, the discrete event reactive control that captures the redundancy management requires a tool that facilitates modelling state transition behaviour. It is preferable to represent the hydraulic actuators and the mechanics of the elevator, on the other hand, in continuous-time models. Stateflow, Simulink and SimMechanics were selected as the respective tools because they facilitate the connection of the different models. The top level of the plant model contains an actuators subsystem that consists of models of the four hydraulic actuators. Each elevator is connected to two actuators by the force exerted by the piston of the actuator cylinders. The dynamic model representing the mechanics of the elevators is shown in *Figure 3. Here the elevators are modelled using blocks from the SimMechanics library. Note the recognizable correspondence between the model representation and the physical representation of the elevator system. The elevator model clearly shows the bodies, (characterized by mass and inertia), being connected to other bodies via joints. With the joints defining the constraints of one body's motion relative to another (e.g.: rotation). The main reasons for using SimMechanics to represent the mechanical dynamics of the elevator were:

a.) It eliminated the need to derive from first principles the mechanical equations of motion of the device

b.) Its sensor and actuator blocks allowed standard Simulink signals to actuate the bodies and to measure the bodies' response.

The piston force used to actuate the elevator mechanical model, was provided by the hydraulic system model shown in *Figure 4.

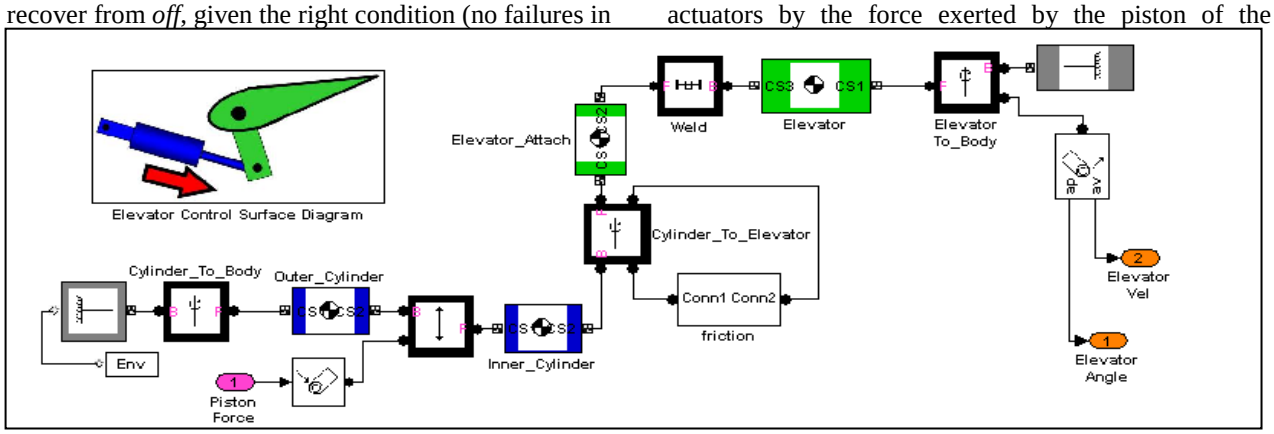


*Figure 3: SimMechanics model of elevator and linkage

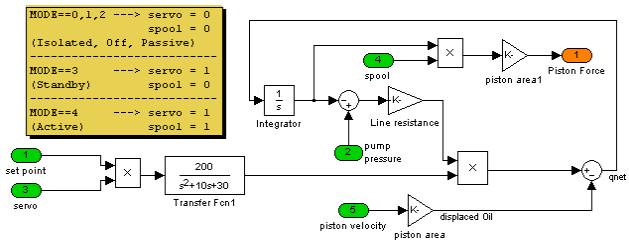


*Figure 4: Hydraulic cylinder model

The actuators consist of a servo valve, a spool valve, and a cylinder, as illustrated in *Figure 4. The servo valve controls the flow of oil into the cylinder as determined by the feedback control algorithm. When the actuator operates in *standby* mode, the piston in the servo valve shadows the movement of the active actuator. To prevent the control pressure in the standby actuator from operating on the cylinder, its spool valve is switched to disallow a flow of oil between the servo valve and cylinder. In this state the actuator is not active, and the spool valve acts as a load to the cylinder, preventing interference with the control exerted by the redundant actuator. Similar to the spool valve, the servo valve can be turned off. When the actuator is in the *passive*, *off* or *isolated* modes, the spool valve does not shadow the control commands, and so the piston in the servo valve is kept at a fixed position. The feedback control law then takes as input the actuator deflection and produces the setpoint of the servo valve piston. The reactive supervisory control that includes the redundancy management described in Section 2 takes the failure status of the elevator position and velocity measurements and produces as output the state of the servo valve and spool valve.

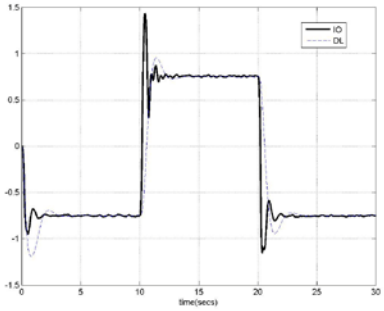


*Figure 5: Elevator step response

Given the models of the hydraulic and mechanical parts of the elevator system, the performance of the different control laws (IO vs. DL) with respect to the switching transients can now be studied. Firstly, a simulation was performed using only the IO control law, and the elevator response to a command step input noted. The same simulation was then repeated for the case of only using the DL control law. The performance of both of the control laws can be seen in *Figure 5 where it can be observed that the DL control law offers less overshoot at the expense of a slightly longer rise time.

To study the effects on the elevator response caused by a sudden change in control law (e.g.; from IO to DL) and actuator set (e.g.; from outer to inner), a simulation was performed that included the representation of hydraulic line failures. At t = 15 [s] a failure of the hydraulic circuit 1 (see *Figure 1) was introduced. The failure causes the actuator control to switch from IO to DL, and the active actuator to switch from the outer unit (primary) to inner unit (backup), for the left actuator.

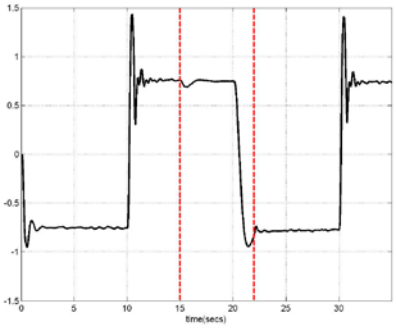


*Figure 6: Elevator step response with standby

The elevator response to this switch in control law is shown in *Figure 6—notice the minor transient effect occurring after the switching event. In the same simulation, a recovery event was injected at t = 22 [s], causing the control law to switch from the DL law back to the IO law—notice from *Figure 6 that this switching event occurs during the settling time phase of the DL control law step response. Although there are some transient dynamics following this switching event, the transition in control law and actuator has a minor impact on the ability of the elevator to track a step input. This robustness is largely because of the design whereby the redundant (backup) actuator is in constant *standby* mode, i.e.; the redundant actuator that was switched to become active was shadowing the actuator that became deactivated.

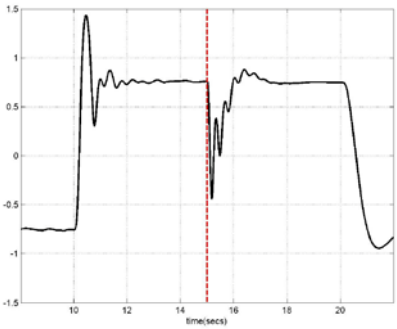


*Figure 7: Elevator step response without standby

Therefore, the piston in the servo valve had no initialization time, and it immediately produced the correct output force when the switch occurred. The significance of this standby design can be highlighted by performing the fault injection scenario for the case where the redundant actuator is configured to have only a passive and active mode of operation (i.e.; in *Figure 2 delete the LI.L1.standby state). Under these circumstances, when an outer hydraulic actuator fault is injected at t= 15 [s], the elevator step response is as shown in *Figure 7—notice the severe dynamics associated in transitioning to the inner actuator and DL control law. Studying such transient effects on the subsystem level, as described above, and at the integrated system level (as presented in other work [6])

are key to Model-Based Design. Model-Based Design allows verification of dynamic behaviour in different scenarios using models, removing the need for a more expensive implementation, such as a physical prototype

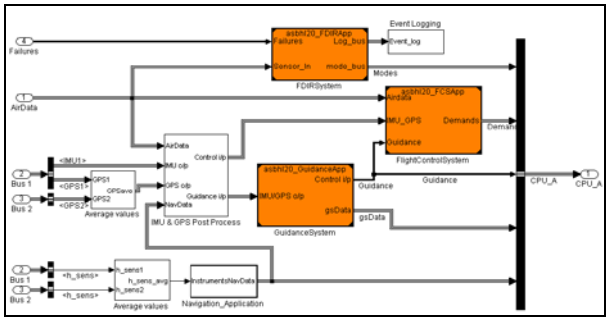

*Figure 8: Integrating the elevator redundancy management into the HL20 model

## 4. INTEGRATING THE REDUNDANCY MANAGEMENT SYSTEM INTO THE HL-20 MODEL

Once the elevator control system has been studied in detail, the effect of failures on overall behaviour and the interaction with other control laws must be studied. This requires the integration of the control law into a more complete aircraft model. In other work [8] an aircraft model of the HL-20 crew rescue vehicle (see *Figure 9) was designed without failure-handling capabilities. The use of Simulink both for modelling the HL-20 as well as the redundancy management control, allows for a straightforward integration of the two. In particular, the redundancy management is included in the aircraft model by reference. This integration is illustrated in *Figure 8**Error! Reference source not found.**, where the model block **FDIRSystem** is a reference to the entire fault detection, isolation, and reconfiguration functionality of the HL-20 primary attitude control systems. Part of this functionality is the redundancy management discussed in Section 2. Other major systems that are referenced are the **FlightControlSystem** and **GuidanceSystem**. Referencing models has a number of distinct advantages:

- The referencing of another model provides a powerful means for model exchange.
  - The referenced models can be independently developed as stand-alone applications and tested to verify that they satisfy their requirements.
  - The referenced models can be directly integrated into configuration management systems, keeping them under version control and allowing configuration management.
- Modelling scalability is enhanced.
  - Models can reference multiple models and can be referenced multiple times themselves.
  - Referenced models can be compiled into shared binaries so they require minimal computing resources.
- Design components can be created with well defined interfaces, guaranteeing that the reference to the model will not modify behavioural characteristics compared to the stand-alone model.
- A test harness can be developed separately from the model by including the models by reference. The same model can then be exploited in different configurations and for different purposes. This streamlines the development process and prevents delays and problems caused by the use of outdated model versions.

*Figure 9 shows a frame of the HL-20 on final approach that is animated using the simulation model described. The position and orientation of the HL-20 as computed by Simulink are visualized using FlightGear [9] and the Simulink to FlightGear interface that is provided by Aerospace Blockset [13].

## 5. CODE GENERATION

Throughout the development of an embedded control system, code is generated for several different applications, such as simulation acceleration, software-in-the-loop simulation, processor-in-the-loop simulation, hardware-in-the-loop simulation, and rapid prototyping [12]. Typically, these applications rely on different hardware platforms, ranging from a desktop personal computer (PC) to fixed-point embedded processors, and they require different optimizations (e.g., static memory allocation instead of dynamic allocation to reduce response times).

*Figure 9: FlightGear animation of HL-20

To support model reuse, it is advantageous if the code for these different applications can be generated from the same model. Code generation then becomes a model transformation process, and by specifying different transformations, the different applications can be supported. This approach enables systems engineers and software engineers to work with the same model. The systems engineers can design the integration of the parts and account for characteristics such as control law stability and response time while the software engineers, using the same model, take care of the scheduling, software partitioning, and data typing attributes.

There are many definitions of 'real-time'. In this paper, real-time is defined so that the simulated behaviour is aligned with the passing of actual, chronological time. In particular, given the sample rate with which the code is executed, the computation of the output from the input must complete in less time than the time between

samples. This implies that the computations cannot be iterative without a fixed upper bound. As such, the model for which code is generated should apply a fixed-step numerical solver and cannot include algebraic loops. Algebraic loops are cyclic dependencies in computations in which the computation of a variable requires its value to be known. Simulink can minimize the occurrence of such cases. If these preconditions are satisfied, real-time amenable code can be automatically generated from a Simulink model by Real-Time Workshop® Embedded Coder [11]. In the redundancy management example in Section 2, real-time C code was generated from the Stateflow chart. A portion of the generated C-code showing the transition tests out of the *passive* mode is shown in Figure 10.

```
case ModeSys_IN_passive:
/* Requirements for passive:
 *  1. Passive mode */
if((ModeSys_DWork.ModeLogic.is_L1_1 ==
    ModeSys_IN_active) != 0) {
 /* Requirements for [LI_act()]:
  * 1. Left Outer Actuator Passive to Standby*/
   ModeSys_DWork.ModeLogic.is_L1 =
   (uint8_T) ModeSys_IN_standby;
   ModeSys_B.LO_mode = 3;
} else
if ((!((ModeSys_DWork.ModeLogic.is_L1_1 ==
       ModeSys_IN_active) != 0)) ||
      ((ModeSys_DWork.ModeLogic.is_L1_0 ==
        ModeSys_IN_active) != 0)) {
 /* Requirements for [!LI_act()|RO_act()]:
  * 1. Left Outer Actuator Passive to Active */
   ModeSys_DWork.ModeLogic.is_L1 =
   (uint8_T)ModeSys_IN_active;
   ModeSys_B.LO_mode = 4;
}
break;
```

*Figure 10: Code generated from a Stateflow® chart

## 6. CONCLUSIONS

Computational models have consistently proven their value in all facets of system design. To fully realize their potential, however, they must be used for all key development tasks, including requirements capture, documentation, code generation, test, and verification. Model-Based Design supports this approach by enabling different design teams to use the same model by elaborating it throughout the development process.

This paper described the use of Model-Based Design in the analysis of fault scenarios in aerospace. It showed how the discrete-event redundancy management system of an elevator control system can be designed and then coupled with continuous-time models that capture the hydraulics and mechanics of the actuator. The resultant hybrid dynamic system can be used to study the transient effects of mode switching.

Model referencing features were then applied to the redundancy management functionality in a more extensive model of the HL-20 crew rescue vehicle. Including the redundancy management functionality in an aircraft model facilitated the use of simulation for training by enabling safe experimentation with failure scenarios. This paper discussed how C/C++ code can be automatically generated in different forms and executed in real time, enabling the redundancy management system to be incorporated with existing flight simulation software.



## ACKNOWLEDGMENTS

The authors thank Rob Aberg and Stacey Gage for integrating the elevator control subsystem into the HL-20 model and for providing the interface to FlightGear. The authors also thank Steve Miller for his contributions in the development of the elevator mechanics model.



## REFERENCES

1 SolidWorks®, *Introducing SolidWorks, SolidWorks*, Concord, MA, 2002.

2 Simulink®, *Using Simulink*, The MathWorks, Natick, MA, March 2007.

3 SimMechanics, *SimMechanics User's Guide*, The MathWorks, Natick, MA, March 2007.

4 MATLAB®, *MATLAB 7 Programming*, The MathWorks, Natick, MA, March 2007.

5 de Lara, J. and Vangheluwe, H., "Defining Visual Notations and Their Manipulation through Meta-Modelling and Graph Transformation," *Journal of Visual Languages and Computing*, Vol. 15, No. 3, June 2004.

6 Mosterman, P. J., Remelhe, M. A. P., Engell, S., and Otter, M., "Simulation for Analysis of Aircraft Elevator Feedback and Redundancy Control," *Modelling, Analysis, and Design of Hybrid Systems*, edited by S. Engell, G. Frehse, and E. Schnieder, Springer-Verlag, Berlin, 2002, pp. 369–390.

7 Mosterman, P. J. and Ghidella, J., "Model Reuse for the Training of Fault Scenarios in Aerospace," *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, Providence, RI, Aug. 2004, CD-ROM, ID: 2004-4931.

8 Gage, S., "NASA HL-20 Lifting Body Airframe Modeled with Simulink and the Aerospace Blockset," *MATLAB Digest* , Vol. 10, No. 4, July 2002.

9 FlightGear, FlightGear is an open-source, multi-platform flight simulator, *http://www.flightgear.org*, 2007.

10 Stateflow®, *Stateflow User's Guide*, The MathWorks, Natick, MA, June 2004.

11 Real-Time Workshop® Embedded Coder, *Real-Time Workshop Embedded Coder User's Guide*, The MathWorks, Natick, MA, March 2007.

12 Mosterman, P. J., Prabhu, S., and Erkkinen, T., "An Industrial Embedded Control System Design Process," *in Proceedings of The Inaugural CDEN Design Conference (CDEN'04)*, July 29 - 30, Montreal, Quebec, Canada, 2004.

13 Aerospace Blockset, *Aerospace Blockset Users Guide*, The MathWorks, Natick, MA, March, 2007.

14 Müller-Glaser, K. D., Frick, G., Sax, E., and Kühl, M., "Multiparadigm Modeling in Embedded Systems Design", *IEEE Transactions on Control Systems Technology*, Vol. 12, No. 2 (March), 2004, pp. 279-292.