Simulation for Analysis of Aircraft Elevator Feedback and Redundancy Control

Pieter J. Mosterman[1], Manuel A. Pereira Remelhe[2], Sebastian Engell[2], and Martin Otter[1]

[1] Institute of Robotics and Mechatronics, DLR Oberpfaffenhofen, P.O.Box 1116, D-82230 Wessling, Germany
[2] Process Control Laboratory, Department of Chemical Engineering, University of Dortmund, 44221 Dortmund, Germany

Abstract. Safety critical systems such as aircraft require functional and hardware redundancy to achieve prescribed safety levels. Discrete event control is applied to ensure that a safe system configuration is available at all times. Since, at present, formal verification techniques are restricted to models with few continuous states, in this paper, simulation is used to verify that the overall system operates according to the requirements when an actuator failure occurs. The feasibility study to modelling and simulation of complex controlled systems presented here is characterised by (i) a complex object-oriented model of aircraft dynamics, including gravity models, aerodynamics, etc., (ii) the specification of the discrete event redundancy control by a domain specific formalism that includes statecharts, (iii) the usage of energy based hybrid bond graphs to model the dynamics of the hydraulic actuators, (iv) model integration on the model level as well as on the data level, (v) support of DAEs with dynamically changing index and (vi) illustrative simulation results.

1 Introduction

Redundancy is one of the most important techniques to achieve the desired level of safety in systems such as aircraft, nuclear plants, chemical plants, and other safety critical applications. Its basic premise is to include redundant functionality into a system that can be activated when failures of the normal operating components occur and to validate and select normal behaviour (e.g., voting procedures).

1.1 Aircraft Attitude Control

To illustrate the concept, consider the primary (attitude) control surfaces of an aircraft as shown in Fig. 1. The ailerons are used to control roll, the elevators control pitch, and the rudder controls yaw motion. This paper concentrates on the pitch control, performed by the elevators. Each of the elevators is positioned by one of two actuators, the other one operates as a passive load. Discrete-event control embedded on two primary flight control units (PFCU) selects the controlling actuator and ensures that the redundant actuator is

loading. Each PFCU controls one actuator per elevator, so that both elevators can be controlled, even if one PFCU fails completely. The PFCUs also generate the position control signals for the four actuators. Feedback control is used for normal operation whereas feedforward control is applied to single actuators in the case of certain failures. To ensure minimal transient disturbances caused by actuator switching, the loading actuator should be shadowing the control signals ready to switch to a mode where it actively controls the elevator. However, in some extreme cases the actuator may be disengaged, i.e., it is loading but not shadowing. Thus, each PFCU has to decide for two actuators whether an actuator is disengaged, shadowing, or controlling and whether a feedback or a feedforward controller is used for shadowing or controlling. These decisions depend on the mode of the other actuator, the state of the other PFCU and the detected failures. The best possible consistent state configuration of both PFCUs for a given failure situation is achieved by a complex iterative interaction of both PFCUs.

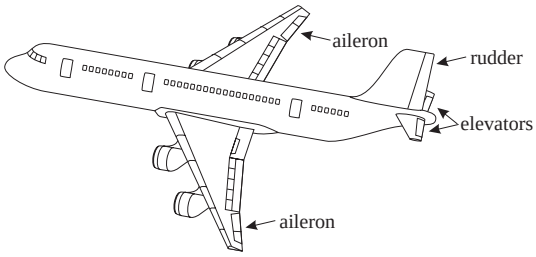


Fig. 1. Primary control surface of an airplane

The hydraulic actuator design and the controller parameters may influence the overall behaviour of the aircraft significantly. Therefore, all contributing parts and phenomena of the aircraft such as aerodynamics, gravity, engines, etc. have to be considered in order to assess the design of the elevator control system. Because of the immense complexity and the intricate redundancy management model-based validation is a necessity.

Formal verification techniques are widely used for pure discrete-event systems and much research has been carried out recently on the verification of hybrid systems. However, at present, the complexity of systems amenable to hybrid systems verification techniques is restricted to a low order continuous dynamics (typically not more than three continuous state variables) [5,13,27]. Consequently, formal methods are applied to the discrete-event part only, e.g., a so-called Failure Mode Effect (FME) analysis is employed to verify certain safety and reliability properties of the redundancy management system. However, its interaction with the continuous parts as well as the design of the position controllers and the hydraulic actuators can not be evaluated

with formal verification techniques. Therefore, the only practical model-based approach for this task is to perform extensive simulation studies.

1.2 Model Design

In this contribution, we concentrate on the modelling and simulation of the elevator control system and the aircraft. The model formulation is driven by the assumption that the simulation studies have the purpose to assess whether the design of the evaluator control system meets the requirements with respect to the overall behaviour of the aircraft (e.g., lateral and longitudinal aircraft velocity and flight path angle). In particular, different sets of parameters of the controllers and of the hydraulic actuators have to be tested in combination with certain failure scenarios.

As a consequence, the simulation model has to incorporate a realistic model of the aircraft dynamics, including all essential effects and components such as aerodynamics, gravity, engines, and hydraulic oil supply. In order to automatically generate the correct boolean input signals of the feedback controllers and the actuators depending on the sequence of failure events it is convenient to include at least the input-output behaviour of the redundancy management components. (In our case it would also be possible to calculate the input signals aforehand, using an external discrete-event simulator, because we do not consider feedback on the redundancy management yet.) Since the sample times of the PFCUs are very fast in comparison to the bandwidth of the actuators, the hardware aspects of the PFCUs can be neglected, i.e., the redundancy management model reacts instantaneously on failures and the controllers are modelled as ideal continuous controllers. Another idealisation can be introduced for the hydraulic actuators. There are many small physical effects such as oil elasticity, viscosity and fluid inertia which do not influence the overall dynamics significantly, but considerably increase the modelling effort, so that these effects are not considered in the corresponding models.

These basic model design decisions cause several difficulties for the modelling and the simulation. With respect to modelling, the complexity of the systems and their heterogeneous nature mandates the use of dedicated formalisms. These formalisms differ greatly in their visual representation and require the interoperation of specific and powerful modelling environments.

Present day simulation technology, on the one hand, can handle large systems of differential and algebraic equations (DAE), possibly extended by some discontinuous equations [2]. On the other hand, discrete-event simulators apply an event driven approach to manage the huge number of state changes in discrete-event models [10]. The combination of discrete and continuous behaviour requires the integration of a numerical integrator with some form of discrete-event simulation. Especially, the detection and location of discrete events during continuous integration has to be supported. Furthermore, at event times discontinuities in continuous state variables may occur.

For the aircraft model, this emerges because the abstractions in the hydraulic actuator models result in a DAE with dynamically changing index. This requires a special simulation engine that switches the active equations and automatically reinitialises the state variables according to physical conservation laws automatically, when the index changes. This contribution presents an approach that copes with all these problems.

1.3 The Modelling and Simulation Approach

For components of the physical system we use object-oriented modelling. In this context, the term object-oriented modelling means that every physical object is modelled independently without making assumptions about its environment and preserving the physical connection structure of the object. The connections of a model component then have to correspond to physical interactions the computational causality of which is not fixed a priori, i.e., the variables involved in an interaction are not a priori defined as inputs or outputs. Furthermore, the behaviour of the component should be defined in a declarative way where a set of (possibly implicit) equations is regarded as a set of behavioural constraints rather than as a calculation formula. To illustrate this, let us consider a hydraulic line. The component model of the line would have two connections which each incorporate a pressure and a flow variable. These variables represent neither inputs nor outputs, since depending on the structure of the environment the pressure drop causes the flow or the flow causes the pressure drop. In some cases the causality can even change dynamically so that a quantity that would be regarded as an input in signal flow diagrams becomes an output and vice versa. This is why the equation-based behavioural description is inherent to object-oriented modelling. Using equations (which may be written in an implicit style) for the description of the behaviour does not impose a specific calculation scheme. From the modelling perspective the equations of all model components and all connections of the overall aircraft model simply form a global set of differential and algebraic equations (DAE) so that simulation is the task to find a solution to these equations, i.e., functions over time that satisfy the equations. To generate efficient simulation code, the model equations must be processed by a symbolic engine and compiled into executable code.

The existing DAE based modelling languages such as gPROMS [4], VHDL-AMS, MODELICA [15], etc. differ in many aspects. This work leverages a library for aircraft models [16,17] developed using MODELICA since it allows building domain-specific graphical component libraries and supports many features known from object-oriented programming such as inheritance, packages, etc. For the modelling and the symbolic processing task DYMOLA [7] was used. It provides a graphical user interface for model composition and symbolic engine of DYMOLA generates C-code from a MODELICA model. Then a standard C-compiler generates the executable simulation code.

This configuration is already powerful enough to model most parts of the aircraft and to simulate the resulting complex DAE system including certain discontinuities (a so-called 'hybrid DAE'). However, for simulating DAEs with dynamically changing index, the symbolic engine of DYMOLA is too limited. Therefore, a specially developed environment, HYBRSIM [22], based on hybrid bond graphs [21] was used to experiment model the aircraft components with variable index, i.e., the hydraulic actuators. The C-code generated by both environments, DYMOLA and HYBRSIM is then merged manually simulated using a general purpose hybrid dynamic system simulator MASIM [20].

The pure discrete-event parts of the elevator control, i.e., the redundancy management, is modelled by a domain-specific formalism including statecharts. It will be shown that the syntax and the semantics of the object-oriented modelling paradigm is not well suited to represent the objects of such a formalism. Instead, a separate modelling environment has been implemented which supports this formalism and generates a monolithic MODELICA component that can be integrated into the MODELICA aircraft model .The behaviour of this component is defined by an algorithm that is taken by MODELICA as an additional model constraint, i.e., it is equivalent to one equations with multiple input and output variables. The essential difference is that the causality of the interface variables and the calculation scheme of the resulting object are predetermined. In contrast to the actuator model that is integrated on the data level, the redundancy management model is integrated on the model level.

Section 2 presents a system level view of the elevator redundancy control. Section 3 discusses the different parts in detail and presents their models. Simulation results are given in Section 4. Finally some conclusions are drawn in Section 5.

2 Aircraft Elevator Control System

The aircraft elevator control system includes several forms of redundancy [26]. The system itself consists of two elevators, the control surfaces. Each of these are controlled by one of two hydraulic actuators while the other one is operating as a passive load. The four actuators take their power from three hydraulic subsystems as depicted in Fig. 2. Two primary flight control units are available to compute actuator control signals and modes.

The functionality of each actuator is specified in textual form in terms of a number of module actuator control modes (MACM) all with their specific behaviour characteristics. These are defined in Table 1. Note that the MACM definitions include behavioural information along with structural information about the particular mode of operation of the actuator components.

The discrete outputs of the redundancy management system are translated into physical behaviour by means of a spool valve and a servo valve

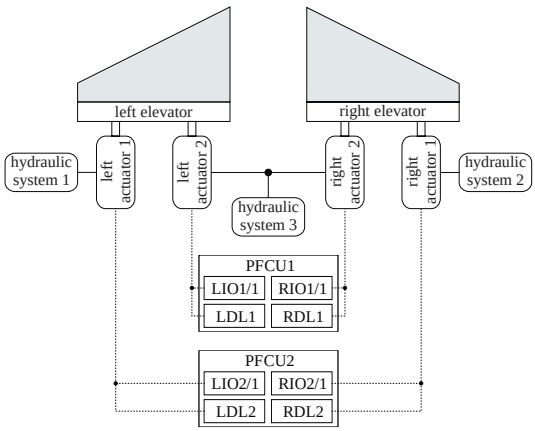


Fig. 2. Elevator system

Table 1. Module actuator control modes

MACM	**Description**	**Actuator**
active	The module controls the servo valve in closed loop mode. The corresponding actuator is active and controls the elevator movement.	servo valve controlled and spool valve open
hot and standby	The module controls the servo valve in closed loop mode. The corresponding actuator is not active and operates as a load.	servo valve controlled and spool valve closed
passive	The module is waiting and does not generate actuator control signals. *It can change its mode at any time to take on control of the corresponding actuator.*	
off	The module is turned off temporarily because of an intermittent failure and does not generate actuator control signals. *As long as the failure has not been fixed, it cannot change to a mode where it controls the corresponding actuator.*	servo valve not controlled and spool valve closed
isolated	The module is turned off indefinitely. *A persistent fault in the control loop of the corresponding system isolates the module and it cannot change to a mode where it controls the corresponding actuator.*	

in the hydraulic actuator. Power is supplied by one of the hydraulic systems and delivered to the actuator cylinder that positions the elevator. This flow of energy is modulated by the servo valve, the modulation is computed by a PID feedback control law. The control signals for the actuators are generated by two *primary flight control units* (PFCU) that can operate as *input-output* modules (IOM) or as *direct-link* modules (DLM) controlled by a switch in control law. The IOMs calculate setpoint values for the actuators based on a PID control algorithm and monitor a number of critical system variables and change between the modes in response. The DLMs allow limited but direct control of the actuators in case the IOMs are not available. The control modules can be in different modes for each of the actuators separately. Moreover, they may control other aircraft actuators as well. In addition, the servo valve may not be controlled and its piston then is in a default position. Also, the spool valve can be turned on and off to switch between active control and passive loading. Continuous feedback control drives the elevator to its desired setpoint, while higher level redundancy management selects the active actuator and the control law to be used.

Interaction between the actuator and the aircraft model consist of forces and moments acting on the elevator that is stiffly connected to the actuator positioning cylinder as well as the pressure generated by the hydraulic systems. Three hydraulic systems supply the oil for the actuators shown in Fig. 1. When a failure occurs, redundancy management switches between actuators and oil supply systems to achieve maximum control.

The behavioural redundancy requirements may be formalised by a set of rules for the redundancy management to switch between module actuator control modes as follows [26]:

1. Mode changes only occur when
   - a system failure is detected,
   - control of an uncontrolled elevator is requested, or
   - one module requests control of both elevators which are controlled by separate modules.
2. One module should be simultaneously in either *active*, *hot*, or *standby* for both elevators as long as possible.
3. If not overruled by the previous specification, the module priority is such that the switching sequence is IOM2/1 $\rightarrow$ IOM1/1 $\rightarrow$ DLM2 $\rightarrow$ DLM1.
4. There is always one and only one module that controls one elevator, i.e., that is *active*.
5. In case of failure of the controlling module, control is assumed by a module that is *hot* or *standby*. If no module is in this mode, the one with highest priority that is *passive* assumes control.
6. A module switches to *hot* when the other module that controls the same elevator, and, therefore, is *active*, belongs to another PFCU and both elevators are controlled by IOMs.

7. A module switches to *standby* when the other module that controls the same elevator, and, therefore, is *active*, belongs to another PFCU and one of the elevators is controlled by a DLM.
8. In case of pressure failure, the 'low pressure' signal only serves for fault classification. It does not cause a direct mode change.
9. In case of 'low pressure' and if a sensor detects elevator positioning system failure, the module switches *off*. The module switches back to *passive* only when no system failure is reported and the 'low pressure' condition does not hold anymore.
10. If 'low pressure' is not reported and the elevator positioning system is reported to fail then the module switches to *isolated*.

To prevent nondeterministic switching, priorities are assigned to the possible transitions. Because of the critical nature of switching to the *isolated* mode to prevent damage to the system, this transition has the highest priority. In addition this causes another module to immediately assume control. This is also desired when, e.g., a pressure loss is detected and the module switches *off*. Therefore, the corresponding transition has second highest priority. Another decision criterion is to allow modules to take over control as quickly as possible. As a result, modes that implement as much control as possible should have highest priority. So, when a module can be switched *active* this should be immediately executed rather than first switching to *standby* if this transition is also enabled. This yields the following priorities:

1. transition to *isolated*
2. transition to *off*
3. transition to *active*
4. transition to *hot*, *standby*, and *passive*.

Sensors in the elevator control system provide the PFCUs with information about the functioning of the system. In case of abnormal readings, the entire set of measurements is used to infer a particular failure mode. Details of this inference mechanism are beyond the scope of this project. To test the redundancy management, failure mode effect (FME) analysis investigates the availability of the system for several test cases that embody a set of sensor readings:

- Pressure decrease in the hydraulic system (H1, H2, H3)
- Predefined set of failures (F)
  - IO module failure (1, 2)
  - DL module failure (1, 2)
  - Actuator failure (left inner/outer, right inner/outer)

These failures represent abstractions of actual physical phenomena underlying the failure detection. FME is still the most important step in verifying system safety and reliability of discrete-event control [14,24].

The combined discrete redundancy management for two of the four actuators on each of the four modules results in eight redundancy modules. This adds up to a considerable discrete behavioural complexity. Each module consists of six possible local modes and there are eight such modules. Thus, the total number of modes of the redundancy management control is 48. There is always one and only one active state in each of the discrete-event models. But, because of the redundancy specification, each of the models needs to have information about the mode of each of the other ones. This interaction is based on the MACMs and causes logic connections between each of the actuator control modules. Finally, an additional discrete-event model is used to model possible fault scenarios by activating states that correspond to particular failure modes. This model has eleven states.

3 Modelling the Parts of the System

The elevator control system described in Section 2 contains a number of parts that are best captured by different modelling approaches: (i) the aircraft dynamics, (ii) the redundancy control, including control law switching, and (iii) the actuator switching behaviour.

3.1 Aircraft Dynamics

To investigate the effect of actuator switching on the overall flight characteristics such as nick rate (q) and angle of attack (α), an aircraft model is required. The more realistic this model, the better the analysis results hold for the actual implementation of the aircraft design.

Object-Oriented Modelling The design of a realistic aircraft model is a tremendous task that combines several domains within aircraft design such as (i) aerodynamics, (ii) gravity, atmospheric, and wind models, (iii) engine/thrust models, (iv) rigid body models including the effects of fuel consumption, and (v) systems models for primary (attitude) control.

Traditionally, such complex aircraft models are designed in a computer processable format such as, e.g., FORTRAN, and they would be completely integrated with facilities for behaviour generation, e.g., the numerical solver. This, however, renders the models unwieldy, error-prone, and rather costly to design.

Recently, a more structured approach to aircraft modelling has been developed based on object-oriented modelling techniques and the use of libraries of the domain specific components mentioned before [16,17]. Object-oriented modelling techniques rely on the notion of *encapsulation* to hide the details of physical component models and to increase maintainability. Furthermore, the models are organised hierarchically which allows successive refinement of behaviours at increasing levels of detail.

Graphical Syntax Figure 3 shows a top-level view of the aircraft model with the engine objects (left), the systems component (top) and the aerodynamics model (right), the rigid body model and the gravity/atmosphere/wind models (bottom-right). These components can be hierarchically decomposed in similar object diagrams.

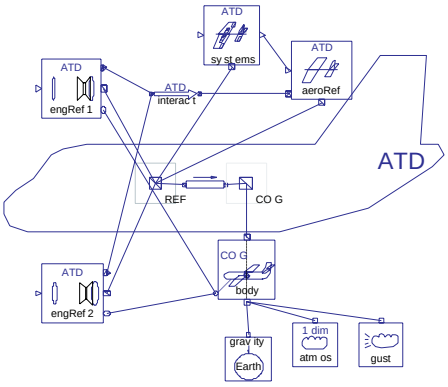


Fig. 3. Top level object diagram of the aircraft model

Communication between objects is through ports that also constitute the interface at the next level in the hierarchical decomposition. For a set of connected variables, v_i, these ports use two different connection semantics, (i) $\forall i(i \neq 0 | v_i = v_0)$, i.e., all connected variables are set equal, and (ii) $\Sigma_i v_i = 0$, i.e., the connected variables are summed to 0. This allows for a convenient implementation of energy flows across ports where the different semantics correspond to the across and through variables, respectively, the product of which constitutes power.

Execution Model The behaviour of each of the primitive model objects is described in terms of algebraic and differential equations. These are treated as noncausal, i.e., no computational direction of the variables is assigned (it is not determined which variable is to be computed from an equation), which is a convenient way of modelling physical systems in terms of declarative constraint specification. Furthermore, it enhances reuse.

To illustrate, consider the elevator control surface library component in Fig. 4. This surface consists of one or more movable parts to adjust the aerodynamic force acting on the aircraft. The library component connects to the remainder of the aircraft model by three ports: (i) a mechanical port *flangeAct*, that contains the elevator deflection, δ, (ii) an aerodynamic port, *deflection*, that carries the forces because of airflow around the elevator, and (iii) a mechanical port, *MOTI*, that contains the force acting on the aircraft.

The elevator computes the force F_{act} from

$$F_{act} = f(V_a^2,\, \delta,\, \rho,\, S_{cs}, \ldots,\, m_{cs},\, g,\, \ldots), \tag{1}$$

the deflection δ from

$$\delta = k_{kin} x_{act}, \tag{2}$$

and, finally, its rate of change $\dot{\delta}$ from

$$\dot{\delta} = k_{kin} v_{act}\ . \tag{3}$$

Note that k_{kin} is a parameter internal to the object that represents the kinematics of the mechanism.

Table 2. ControlSurface class interface variables

Interface Variables		
across	x_{act}	displacement of actuator flange
	v_{act}	displacement velocity of actuator flange
	V_a	airspeed velocity
	ρ	air density
	g	gravitational acceleration
through	F	force acting on actuator flange

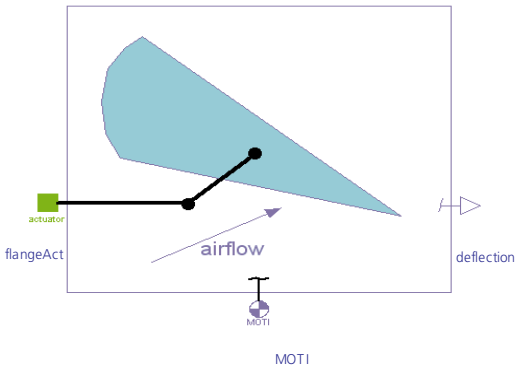


Fig. 4. Elevator control surface library component

To enable execution, the primitive object equations and the connection constraints for across and through variables are accumulated by a global model interpretation scheme. It sorts and solves the overall system of differential and algebraic (DAE) equations by assigning causality so that the

unknowns can be computed from the number of equations and input and state variables. Algebraic manipulations are performed to reduce the system of equations (e.g., [3]).

To represent switching, equations may be conditionally active. When the conditions change their truth value, this causes *events*. When events occur, variables may undergo discontinuous changes. In addition to the differential and algebraic equations, a 'pre' operator is defined to allow access to the value of a variable immediately before a discontinuous change. Because this introduces discrete state behaviour, an iteration is required to converge to a consistent state before the continuous simulation is resumed. Though this mechanism can be used for implementing discrete-event behaviour, it is difficult to mimic state transition diagrams using object diagrams and even more so to describe the state transition behaviour by local equations of the primitive states and transitions. The graphical syntax of object diagrams does not allow annotation of component connections, thus it is not possible to write conditions, events, and actions alongside a transition. Furthermore, transitions are not objects in object diagrams. Therefore, the transition behaviour requires a specific transition object to be inserted. Execution has to be described in terms of local algebraic constraints that communicate between states and transitions to evaluate whether a state is active and a transition is enabled [23].

The result of collecting the local equations, adding the connection constraints, and sorting and solving these leads to a global system of equations of the form

$$\begin{aligned} \dot{x} &= f_\alpha(x, u, t) \\ 0 &= g_\alpha(x, u, t) \\ \alpha^+ &= \phi_\alpha(x, u, t) \end{aligned} \tag{4}$$

where f_α specifies the dynamics in mode α, g_α the event generation functions ('zero crossings'), and ϕ_α the next mode function. Before continuous simulation can start or be resumed after an event occurred, a consistent mode α, i.e., $\alpha^+ = \alpha$, has to be found. Typically, this is performed by a fixed point iteration scheme.

3.2 Redundancy and Position Control

The main purpose of the two primary flight control units is the generation of appropriate continuous and discrete control signals for the four elevator actuators. Each PFCU contains a failure monitoring function, a specific discrete-event part for the redundancy management as well as a switched position controller. When a failure is detected, the redundancy management parts of both PFCUs interact tightly in order to achieve a consistent decision on the appropriate reaction, before switching the operating control laws. This is because each PFCU is responsible for different actuators and has to take

the discrete state of the other PFCU into account in order to guarantee that each elevator is controlled by one actuator only. Therefore a simple failure may trigger a sequence of transitions in both PFCUs, where a discrete mode transition in one PFCU may lead to a state which forces another transition in the other PFCU and so on.

Graphics Since hardware aspects are beyond the scope of this paper, the redundancy management parts of both PFCUs are unified in one discrete-event model component neglecting the distributed architecture of the system. As a consequence, the aircraft model contains only one elevator control component. This is divided into three parts: (i) a failure injection module that replaces the failure monitoring functions allowing to directly study specific failure scenarios, (ii) the combined redundancy management parts of both PFCUs that react on changes of the failure configuration and (iii) the switched position controllers of both PFCUs whose transfer functions depend on the actual modes of the redundancy management component (Fig. 5).

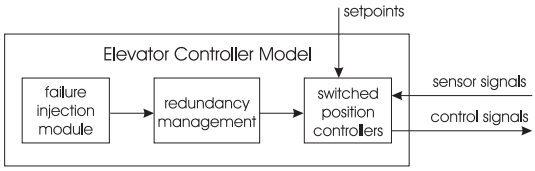


Fig. 5. The structure of the elevator controller model

The requirements for the redundancy management which were formulated informally in Section 2 state that each redundancy module contains 6 possible local modes. Since a redundancy module switches from one mode to another under certain conditions, the modules should be modelled by a kind of state transition diagram, where the modes are represented by discrete states and the transition arrows represent the possible mode switchings. In order to take the transition priorities into consideration, hierarchical states as known from the statechart formalism [11] are used. Additional states are introduced that do not correspond to a mode, but represent the priorities of the transitions: The higher a state in the hierarchy the higher is the priority of its outgoing transitions, e.g., the transition ToOff in Fig. 6 has a higher priority than ToAct and a lower priority than ToIso. The statechart model in Fig. 6 reflects the state transition aspects of a redundancy module declared in the informal description of the requirements.

The transition conditions can be derived from the switching rules of the requirements and differ for each statechart. In order to keep the statechart model generic and take architectural aspects into consideration a specific hierarchical block diagram formalism is used (Fig. 7). Two blocks on the top level represent the two primary flight control units. The input port contains

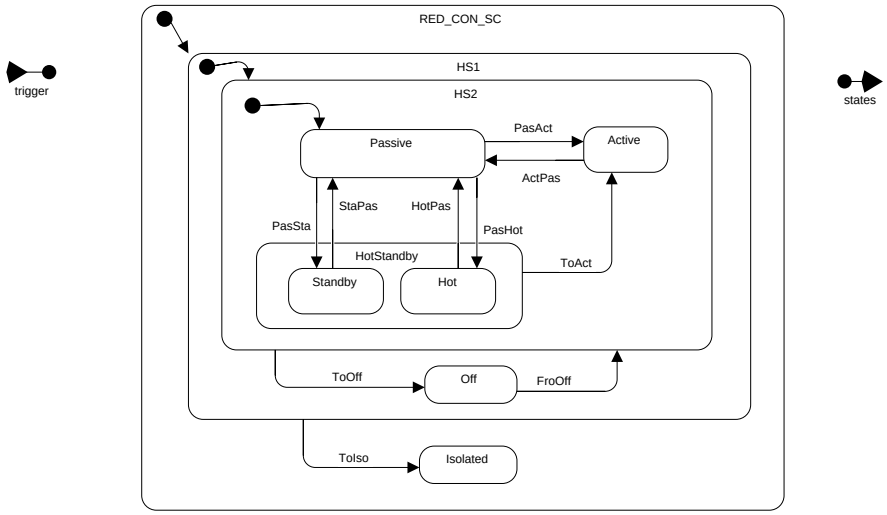


Fig. 6. The redundancy module statechart

the failure values that come from the failure injection module whereas the output ports transmit the actual module modes to the switched controllers. Each PFCU block contains four control modules (LIO, RIO, LDL and RDL) as subblocks the behaviour of which is defined by the statechart in Fig. 6. The transition conditions are calculated outside of the statecharts in a special block (PFCU1_Logic and PFCU2_Logic) with no state behaviour.

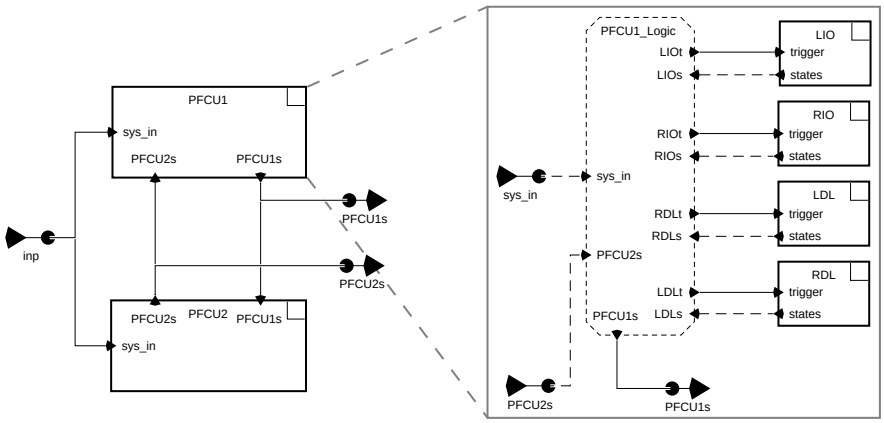


Fig. 7. Two block diagrams of the discrete-event part

Execution The intended behaviour of the elevator control model is as follows: The failure injection module generates Boolean signals that indicate the presence of specific failures. When a failure signal changes, the transition conditions of both PFCUs are evaluated and their values are transmitted to the statechart blocks that perform their transitions independently. After all statecharts have converged to a persistent discrete state, i.e. no further transitions happen, the transition conditions are calculated again taking the new states into account and a new set of transitions may be performed in the modules again. When the overall discrete-event system reaches a persistent state, this local event iteration is stopped and the output values are set, and the position controllers may change their mode.

In order to analyse the behaviour of the elevator control and the overall aircraft for different failure scenarios, the failure injection module generates predetermined sequences of failures. These scenarios can be modelled by equations containing logical expressions and inequalities over the independent variable time and parameters as shown in the following example where IO2failure is present from time t_1 to t_2:

$$\text{IO2failure} = (t > t_1) \wedge (t < t_2). \tag{5}$$

The output of the redundancy management part switches the position controllers that are easily described using equations. The following example shows the controller equations of PFCU1 for the left elevator:

$$e_{act,l1} = w_{act} - x_{act,l} \tag{6}$$

$$u_{act} = \begin{cases} 0 & \text{PFCU1states.LIO.Off} \\ & \vee\ \text{PFCU1states.LIO.Isolated,} \\ w_{act} & \text{PFCU1states.LDL.Active,} \\ k_p e_{act,l1} + k_d v_{act,l} & \text{else} \end{cases} \tag{7}$$

$$u_{spool,l1} = \text{PFCU1states.LIO.Active} \tag{8}$$

Remarks to the modelling of discrete-event systems In the former section the redundancy management part was modelled by a domain specific formalism including statecharts, and the resulting input/output behaviour can be described by an algorithm. There are two alternatives for realising the graphical editor and the integration into the object-oriented aircraft model. The first one (presented above) requires the implementation of a domain-specific graphical editor and a translation procedure that generates code from the models that contains the resulting switching algorithm in an equation-compatible style and that can be inserted as an object into the aircraft model. The second alternative does not require any tool development.

Here the object-oriented approach is applied directly to the modelling of the discrete-event system. This means that basic model objects, such as states and transitions, are defined by equations in a way that the composition of these objects results in a system of equations that behaves corresponding to the desired semantics. The latter approach was successfully applied to simple automata and Petri-net models [23]. However, there are several serious deficiencies so that this approach is not suitable for complex systems.

As far as the graphical representation is concerned, connecting state and transition objects through their ports is somewhat clumsy, since the original formalisms such as automata, Petri-nets and statecharts do neither have ports nor do transitions correspond to graphical nodes. In contrast to automata and Petri-nets, the statechart formalism supports a hierarchy concept that can not be represented in an acceptable way by object diagrams: Due to the concept of encapsulation all objects have to hide their internal objects and interaction across the object hierarchy is not allowed. Therefore it is not possible to view all states of a statechart at the same time. Furthermore, inter-level transitions that cross state levels, i.e. PasHot, HotPas, StaPas and PasSta, cannot be supported.

With respect to the execution semantics there is at least one critical problem: The blocks of the redundancy management model have to interact locally and must reach a consistent state before propagating new values to the rest of the system. Using an object-oriented approach, it is not possible to realise this local event iteration. The reason is that the behavioural description of the eight statechart blocks and the transition condition blocks (PFCU1-Logic, PFCU2-Logic) would be mixed into the global set of equations of the overall system, since each equation has to be treated similarly. Thus, for each calculation of the transition conditions the global set of equations has to be evaluated simultaneously. But this means that an intermediate state of the redundancy management, i.e. an inconsistent mode, would be inevitably linked to the physical system which is not the intended behaviour. The same holds for state transition diagrams or Petri-nets modelled as object diagrams: Each firing of a transition corresponds to a global evaluation of the system of equations of the overall model so that local event iterations can not be implemented.

Because of these problems, we created a tool which supports the first alternative [18]. A graphical modelling environment that supports the block diagram and the statechart formalism was realised. In order to reduce the effort for introducing additional domain specific formalisms, e.g., sequential functions charts, the meta-modelling tool DoME [1,8] is used that allows the formal specification of the syntax and the appearance of graphical formalisms as well as the programming of automatic translation procedures. Figure 6 and Fig. 7 were created by DoME. The translation procedure generates one monolithic Modelica object that contains an algorithm that con-

sists of sequential assignments and simple control flow statements and that corresponds exactly to the desired switching sequence.

3.3 Actuator Dynamics

The hydraulic actuators are the interface between the discrete-event domain of the redundancy control and the continuous domain of the aircraft dynamics. The actuator here is not modelled with all details as this would lead to steep gradients in the behaviour that are difficult to handle and slow down simulation of the aircraft behaviour, even if efficient numerical solvers such as DASSL [25] are used.

Higher Index DAE The necessity to remove small physical effects such as fluid storage in lines and oil elasticity and viscosity leads to DAEs with a higher complexity because state variables are directly coupled instead of interacting through additional states with small time constants. Though these DAEs can be solved by differentiation before simulation starts, the switching effects of the actuators may cause such algebraic constraints to emerge during simulation, requiring two phenomena to be handled: (i) the state variables that become algebraically coupled are constrained to a subspace of reduced dimension and the values before the constraint becomes active have to be projected into this subspace, and (ii) future dynamic behaviour of these state variables must be in this reduced subspace.

To illustrate these notions, consider the actuator model in Fig. 8. When initially the actuator is *active*, the supply path is open, i.e., control signals generated by the servo valve are supplied to the positioning cylinder, causing the piston to accelerate. When, at a given point in time, the actuator is switched to be *off*, the loading path becomes active. Because of the inertial effects in the loading pathway, there is dependency between the piston and this fluid inertia and an algebraic constraint between these two variables ($v_{piston} = -A_p f_{load}$) restricts the state space in which the system evolves. This is illustrated in Fig. 9(a), where the double arrow heads on the dashed field lines indicate the direction of the discontinuous change. This algebraic dependency can be eliminated by introducing small parasitic storage effects for the piping and some oil elasticity and viscosity, but this adds very steep gradients to overall system behaviour as illustrated in Fig. 9(b) that complicate simulation and are not relevant for the overall behaviour of the aircraft.

The implicit jumps in the state variable values have to be computed during simulation. This requires the system of equations to be in general DAE format, $0 = f_\alpha(\dot{x}, x, u, t)$, where the index of f_α may change during simulation and a simulator that facilitates these reinitialisation computations such as MASIM.

At present, commercially available simulation tools cannot handle such abrupt changes in DAE models. Therefore the experimental modelling and

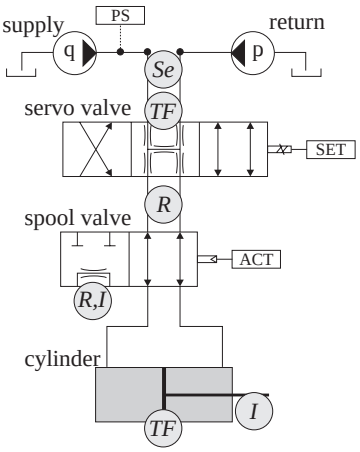
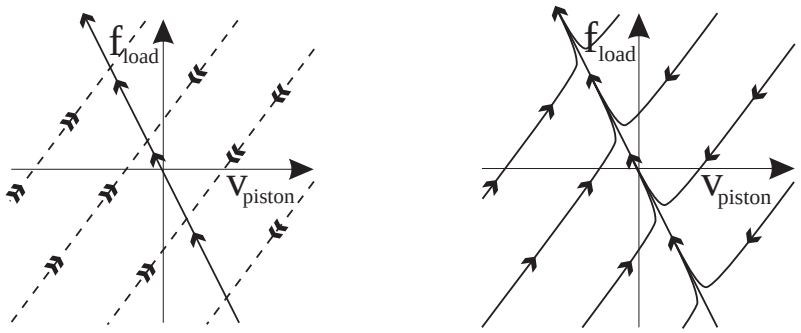


Fig. 8. Schematic of hydraulic actuator



(a) Discontinuous changes

(b) Steep gradients

Fig. 9. Phase space for v_p and f_{load}

simulation environment HyBrSim [22] has been realised for the purpose of testing algorithms for the reinitialisation of switched systems with index changes. HyBrSim is based on bond graph modelling of the physical system.

Bond Graph Model of the Actuators Figure 10 shows the hybrid bond graph model of the two left hydraulic actuators. The two Se elements[1] are sources (inputs) of a bond graph model which are connected to the hydraulic circuits in the aircraft model that provide the input pressure. The servo valve modulation is applied by the *TF* elements, where the *setL1* and *setL2* elements are connected to the setpoint generated by the aircraft control model.

[1] The element type is listed on the left of each element rectangle.

The I elements represent connections (equal flow points) and the attached R element captures dissipative effects. Note that these are modelled as linear phenomena. The *loadL1* (*loadL1*) connection also has some inertia associated with it, embodied by the *IloadL1* (*IloadL2*) element. The cylinder chamber is modelled by a 0 element, an equal pressure point. Both cylinders connect through a piston with area modelled by a *TF* element to one equal velocity point for the elevator control surface movement. This velocity, as well as the displacement and force are inputs to the aircraft model.

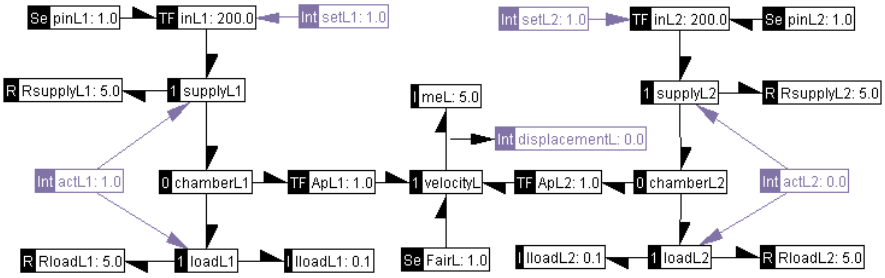
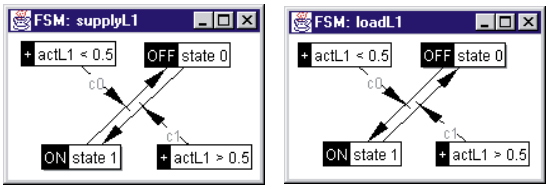


Fig. 10. Hybrid bond graph of the two left hydraulic actuators

The switching behaviour is modelled by two *controlled junctions* [21] in each actuator, in the left actuator these are *supplyL1* and *loadL1*. The local finite state machines that control their states are given in Fig. 11. The control event *actL1* is generated by the redundancy control in the enclosing part of the model. When the *supplyL1* junction is ON and *loadL1* is OFF, the actuator is active. When *supplyL1* is OFF and *loadL1* is ON, it is loading (either hot, standby, passive, or isolated). Note that the mutual switching constraints allow no other configurations.



(a) Supply path (b) Load path

Fig. 11. Finite state machines of actuator 1 in the hybrid bond graph

Equations The equations generated from the hybrid bond graph by HYBRSIM incorporate the switching effect as guarded equations. This prevents the need for pre-enumeration which would cause an exponential growth of the number of modes.[2] For example, for the loading pathway, *loadL1*, the equation generated is

$$0 = (-chamberL1.p + IloadL1.p + RloadL1.p)\alpha_i + (loadL1.f) * (1 - \alpha_i) \quad (9)$$

where α_i is the i^{th} entry in the mode vector α. This ensures that in a mode where this connection is active, $\alpha_i = 1$, the pressure drops of the connected elements are balanced. When the connector is not active, $\alpha_i = 0$, the fluid flow through *loadL1* becomes 0. This models ideal switching but may lead to higher index DAEs (e.g., because *IloadL1* and *mpL* become algebraically related). A numerical solver such as DASSL can handle systems up to index 1 directly and up to index 2 with some provisions, e.g., the step-size control of index 2 variables needs to be switched off [6]. Another prerequisite is that DASSL should be given a set of consistent initial conditions, i.e., those that are in the subspace of continuous behaviour. This is achieved by applying a projection mechanism which is consistent with physical conservation laws [9,28,29].

The discontinuous changes are computed by first linearising the system with a finite difference method. Then a pseudo Weierstrass normal form is derived (up till index 2)

$$0 = \begin{bmatrix} \bar{E}_{11} & 0 & 0 \\ 0 & 0 & \bar{E}_{22,12} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\bar{x}}_1 \\ \dot{\bar{x}}_{2,1} \\ \dot{\bar{x}}_{2,2} \end{bmatrix} + \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12,1} & \bar{A}_{12,2} \\ 0 & \bar{A}_{22,11} & \bar{A}_{22,12} \\ 0 & 0 & \bar{A}_{22,22} \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \bar{x}_{2,1} \\ \bar{x}_{2,2} \end{bmatrix} + \begin{bmatrix} \bar{B}_1 \\ \bar{B}_{2,1} \\ \bar{B}_{2,2} \end{bmatrix} \begin{bmatrix} u \end{bmatrix} \quad (10)$$

where $\bar{E}_{11,11}$, $\bar{A}_{22,11}$, and $\bar{A}_{22,22}$ are of full rank. This allows computation of the initial conditions as [19]

$$\begin{aligned} \bar{x}_1 &= \bar{x}_1^0 + \bar{E}_{11}^{-1}\bar{A}_{12,1}\bar{A}_{22,11}^{-1}\bar{E}_{22,12}(\bar{x}_{2,2} - \bar{x}_{2,2}^0) \\ \bar{x}_{2,1} &= -\bar{A}_{22,11}^{-1}(\bar{B}_{2,1}u + \bar{E}_{22,12}\dot{\bar{x}}_{2,2} + \bar{A}_{22,12}\bar{x}_{2,2}) \\ \bar{x}_{2,2} &= -\bar{A}_{22,22}^{-1}\bar{B}_{2,2}u \end{aligned} \quad (11)$$

where $\bar{x}_0$ are the user-provided initial values after the coordinate transformation to achieve the desired normal form, $\bar{x}_0 = Zx_0$. The values for $\bar{x}$ can then be transformed back to obtain initial values for x that are consistent with the subspace of the dynamic behaviour, and thus the implicit jump is determined.

4 Simulation of the overall system

The aircraft model, the redundancy control system, and the actuator feedback and discrete event control were modelled using different modelling formalisms

[2] For the hybrid bond graph in Fig. 10 there are already $2^4 = 16$ modes, although only two occur during normal operation.

and tools (DYMOLA, HYBRSIM,DOME). Each of these is best suited for the respective task.

4.1 Integrating the Components

To enable a comprehensive analysis, however, the parts have to be integrated into a coherent model. This can be achieved at two integration levels: (i) at the model level, and (ii) at the data level. In the first case, a model of a component is embedded into the overall model using a common formalism. In the second case two (or more) distinct simulators are generated and connected together such that an appropriate data exchange is organized during simulation.

Since the descriptions of the failure injection module and the redundancy management system laws are based on equations, they can be incorporated easily into the object-oriented and equation-based aircraft model. This also holds for the hydraulic actuators, in principle, because the bond graph models correspond to a set of hybrid differential and algebraic equations. But due to present restrictions of the simulation software available for object-oriented modelling languages, specific simulation code is generated from the bond graphs of the actuators and merged with the simulation code that results from the aircraft model. Thus, the failure injection module and the switched control laws are integrated with the aircraft model at the model level, whereas the actuators are integrated at the data level.

For the redundancy management component a mixed integration approach is used. On the one hand the modelling environment for the discrete-event parts generates a simulation algorithm that defines the input-output behaviour of the discrete-event component which is a data level specification. On the other hand, this automatically generated algorithm is designed in a way that is compatible to the MODELICA language so that it can be embedded directly into the aircraft model. In MODELICA such an algorithm is regarded simply as an additional model constraint that corresponds to a set of equations with a fixed set of input and output variables.

To simulate the resulting hybrid model, MODELICA's hybrid DAE semantics is exploited. The temporal inequality expressions in the failure injection module are transformed into time events for the numerical integrator so that the continuous integration stops exactly when a switching time has elapsed. Then the whole set of equations is re-evaluated with the new values of the inequality expressions. Thereby, the algorithm of the redundancy management is also re-evaluated resulting possibly in a new state which may switch the feedback control laws.

4.2 Simulation Results

The phugoid in Fig. 12 is the result of two interacting phenomena: When the aircraft pitch angle increases, it gains altitude and at the same time loses airspeed. Because of this loss of airspeed, there is less upward thrust,

which causes the aircraft to lose altitude in return. However, as it starts losing altitude, it picks up speed again and the airspeed rises. This results in a slightly damped oscillatory behaviour which is required to be stable in commercial aircrafts.

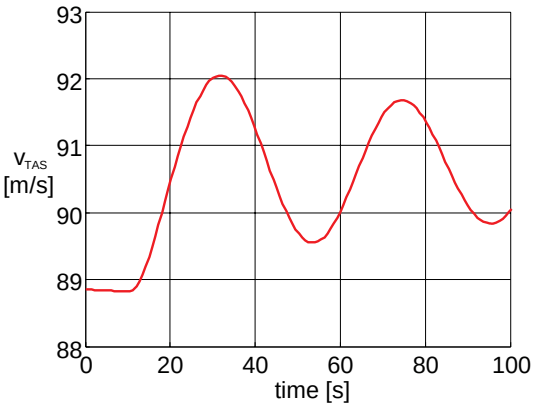


Fig. 12. Simulation shows a phugoid typical for aircraft

To investigate the effect of the redundancy control on the aircraft's behaviour, an actuator failure is introduced during a setpoint change. The setpoint change occurs at $t = 0.05$ $[s]$ and the actuator failure at $t = 0.08$ $[s]$. Figure 13 shows that the failure leads to an immediate change of the active actuators and the switching transients in the hydraulics cause a sharp drop in elevator velocity. Because small effects such as oil elasticity and viscosity are neglected in the simulation, this results in a discontinuous change that occurs because of the algebraic dependency between elevator inertia and fluid inertia of the new loading path.

During a short period of time, the PID control causes the elevator velocity to ramp up to the value which it would have assumed without the failure. Note the short delay that is possible because the actuator that switches to active was *hot* and *shadowing* the PID control.

The aircraft redundancy control is designed such that an actuator failure should not have a noticeable effect on the behaviour of the aircraft. Using the comprehensive model with switching logic and transients, and an extensive model of the aircraft dynamics, this effect can be studied as well. Figure 14(b) shows the effect of the actuator switch on the aircraft pitch angle, and Fig. 15(b) shows the effect on the pitch angle velocity. This verifies that the actuator switch has almost no effect on the overall aircraft behaviour which, because of the realistic aircraft model, provides much confidence for the real implementation. Note that the (small) effect of the actuator switching on global behavior manifests itself after a significant delay.

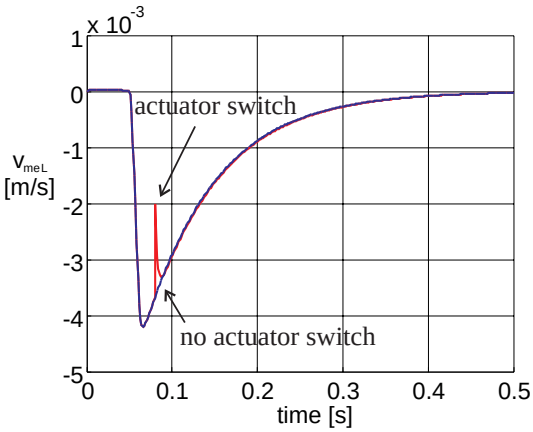
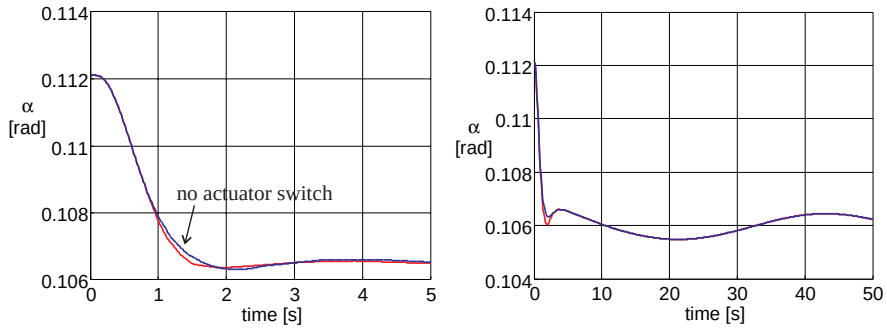


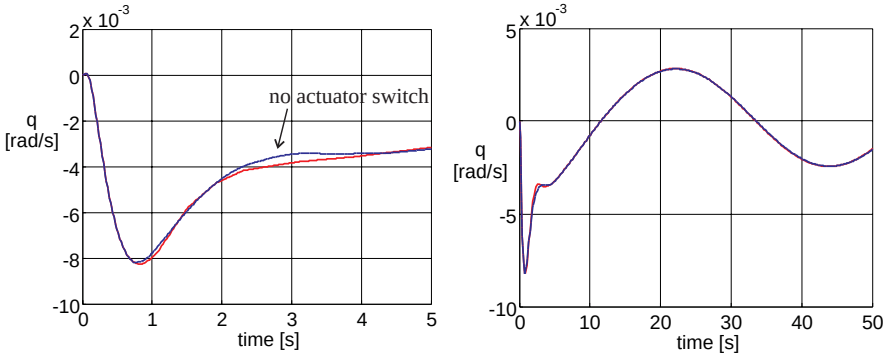
Fig. 13. Elevator velocity when a failure occurs at t = 0.08 shortly after a setpoint change at t = 0.05



(a) Detailed view (b) Overall behaviour

Fig. 14. Pitch angle for normal behaviour and for an actuator switch at 0.08 [*s*]

Table 3 illustrates how the redundancy management reacts, when the IO module failure occurs in PFCU2. In the first local transition the statecharts of LIO and RIO (Left / Right IO) of PFCU2 switch from *Active* to *Isolated*, since this modules should not be activated again (see rules 1 and 10 in Section 2). Then PFCU1 takes over the actuators by activating its LIO and RIO modules (rules 1, 3, 5). In the last local transition, the LDL and RDL (Left / Right DL) statecharts of PFCU2 switch into the *Hot* mode preparing the system for a possible second failure (rule 6). Since state 2 would violate rule 4 and the transition from state 3 to state 4 would violate rule 1, the internal iterations have to be hidden from the outer system in order to prevent inconsistent



(a) Detailed view (b) Overall behaviour

Fig. 15. Pitch angle velocity for normal behaviour and for an actuator switch at 0.08 $[s]$

outputs. This is why only the global transition from state 1 to state 4 is made observable to the outside.

Table 3. State transitions of the redundancy management system

local steps		1	2	3	4
PFCU2	IO	Active/Active	Isolated/Isolated	Isolated/Isolated	Isolated/Isolated
	DL	Passive/Passive	Passive/Passive	Passive/Passive	Hot/Hot
PFCU1	IO	Hot/Hot	Hot/Hot	Active/Active	Active/Active
	DL	Passive/Passive	Passive/Passive	Passive/Passive	Passive/Passive
actuator	outer	control/control	-/-	-/-	shadow/shadow
	inner	shadow/shadow	-/-	-/-	control/control
global visibility		yes	no	no	yes

5 Conclusions

The comprehensive model of the aircraft developed here incorporates the redundancy management system, the switched positioning controllers, the actuator models as well as a complex model of the general dynamics of the aircraft. Hence, it is possible to assess the design of the elevator control system with respect to the overall behaviour of the aircraft in the case of failures. Since the less important physical effects of the hydraulic actuators were neglected, the simulation is fast enough to be used also in the context of a multi-objective parameter optimisation (MOPS) [12]. Such an optimisation may, e.g., reduce the elevator surface or the actuator power such that the switching transients still do not affect the level of aircraft handling.

The abstractions used in the actuator models, i.e. neglecting small physical effects such as oil elasticity and viscosity, result in a DAE that may change its index during simulation. A standard DAE solver, such as DASSL, can be applied for this model, if the re-initialisation at event times results in a consistent state. For a correct behavioural simulation, this re-initialisation has to satisfy the physical conservation laws. For the purpose of this feasibility study the actuators were modelled in HyBrSim, a modelling environment based on hybrid bond graphs that supports the necessary re-initialisation procedure. The C-code generated by this environment was manually combined with the C-code generated by Dymola which includes the rest of the aircraft model. The hybrid system simulator MAsim was used to generate behaviors. MAsim has facilities to compute discontinuous changes of generalized state variables as algebraic constraints between them become active. The discrete-event parts of the aircraft are modelled using a visual specification language and are translated into a Modelica algorithm that can be integrated into the aircraft model on the model level [18].

The presented modelling and simulation approach that combines an object-oriented modelling language such as Modelica, domain-specific model libraries, discrete-event modelling formalisms and powerful simulation methods including correct state re-initialisation, was successfully applied to the aircraft elevator control system and seems to be promising for general complex technological systems.

References

1. DoME guide. http://www.htc.honeywell.com/dome/, Honeywell Technology Center, Honeywell, 1999. version 5.2.1.
2. ABACUSS. http://yoric.mit.edu/abacuss/abacuss.html, 1995. Massachussets Institute of Technology.
3. M. Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems.* PhD dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1994.
4. P. I. Barton. *The Modelling and Simulation of Combined Discrete/Continuous Processes.* PhD dissertation, University of London, 1992.
5. Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*. Springer-Verlag, March 2001.
6. Pawel Bujakiewicz. *Maximum weighted matching for high index differential algebraic equations.* PhD dissertation, TU Delft, Delft, Netherlands, 1994. ISBN 90-9007240-3.
7. Dymola. Homepage: http://www.dynasim.se/.
8. Eric Engstrom and Jonathan Krueger. A meta-modeler's job is never done: Building and evolving domain-specific tools with DOME. In *Proceedings of the IEEE International Symposium on Computer Aided Control System Design*, pages 83–88, Anchorage, Alaska, September 2000.

9. Eberhard Griepentrog and Roswitha März. *Differential-Algebraic Equations and Their Numerical Treatment*. BSB Teubner, Leipzig, 1986. ISBN 3-322-00343-4.
10. IEEE 1076.1 Working Group. IEEE standard 1076.1-1999, March 1999. http://www.vhdl.org.
11. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
12. Hans-Dieter Joos. A methodology for multi-objective design assessment and flight control synthesis tuning. *Aerospace Science and Technology*, 3(3):161–176, 1999.
13. Nancy Lynch and Bruce Krogh, editors. *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*. Springer-Verlag, March 2000.
14. G. Mai and M. Schröder. Simulation of a Flight Control Systems' Redundancy Management System using Statemate. 7. User group meeting STATEMATE, April 1999.
15. Modelica. Homepage: http://www.modelica.org/.
16. D. Moormann, P.J. Mosterman, and G.-J. Looye. Object-Oriented Computational Model Building of Aircraft Flight Dynamics and Systems. *Aerospace Science and Technology*, 3:115–126, 1999.
17. Dieter Moormann. *Automatisierte Modellbildung der Flugsystemdynamik*. PhD dissertation, Aachen Technical University (RWTH Aachen), Aachen, Germany, 2001. in German.
18. P. Mosterman, M. Pereira Remelhe, S. Engell, and M. Otter. Simulation for analysis of aircraft elevator feedback and redundancy control. In S. Engell, G. Frehse, and E. Schnieder, editors, *Modelling, Analysis, and Design of Hybrid Systems*. Springer-Verlag, 2002.
19. Pieter J. Mosterman. Implicit modeling and simulation of discontinuities in physical system models. In S. Engell, S. Kowalewski, and J. Zaytoon, editors, *The 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems*, pages 35–40, 2000.
20. Pieter J. Mosterman. MAsim. Technical Report DLR-IB-, DLR Oberpfaffenhofen, Oberpfaffenhofen, Germany, 2001.
21. Pieter J. Mosterman and Gautam Biswas. Modeling discontinuous behavior with hybrid bond graphs. In *1995 International Workshop on Qualitative Reasoning*, pages 139–147, Amsterdam, May 1995. University of Amsterdam.
22. Pieter J. Mosterman and Gautam Biswas. A java implementation of an environment for hybrid modeling and simulation of physical systems. In *International Conference on Bond Graph Modeling (ICBGM '99)*, pages 157–162, San Francisco, January 1999.
23. P.J. Mosterman, M. Otter, and H. Elmqvist. Modeling Petri Nets as Local Constraint Equations for Hybrid Systems Using Modelica. In *Proceedings of SCS Summer Simulation Conference*, pages 314–319, Reno, Nevada, July 1998.
24. Stephen Osder. Practical view of redundancy management application and theory. *Journal of Guidance, Control, and Dynamics*, 22(1):12–21, January-February 1999.
25. Linda R. Petzold. A description of DASSL: A differential/algebraic system solver. Technical Report SAND82-8637, Sandia National Laboratories, Livermore, California, 1982.

26. J. Seebeck. *Modellierung der Redundanzverwaltung von Flugzeugen am Beispiel des ATD durch Petrinetze und Umsetzung der Schaltlogik in C-Code zur Simulationssteuerung.* Diplomarbeit, Arbeitsbereich Flugzeugsystemtechnik, Technische Universität Hamburg-Harburg, 1998.
27. Frits W. Vaandrager and Jan H. van Schuppen, editors. *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science.* Springer-Verlag, March 1999.
28. A. J. van der Schaft and J. M. Schumacher. The complementary-slackness of hybrid systems. *Math. Contr. Signals Syst.*, (9):266–301, 1996.
29. George C. Verghese, Bernard C. Lévy, and Thomas Kailath. A generalized state-space for singular systems. *IEEE Transactions on Automatic Control*, 26(4):811–831, August 1981.