

Towards a Unified Model-Based Safety Assessment^{*}

Thomas Peikenkamp¹, Antonella Cavallo², Laura Valacca³, Eckard Böde¹,
Matthias Pretzer¹, and E. Moritz Hahn¹

¹ Kuratorium OFFIS e.V., Escherweg 2, 26121 Oldenburg, Germany
{peikenkamp, boede, pretzer, ernst.hahn}@offis.de

² Alenia Aeronautica S.p.A., Strada Malanghero 17, IT-10072 Caselle, Turin, Italy
acavallo@aeronautica.alenia.it

³ Societa' Italiana Avionica S.p.A, Strada Antica di Collegno 253, IT-10146 Turin,
Italy
valacca@sia-av.it

Abstract. The increase of complexity in aircraft systems demands for enhanced analysis techniques. Methods are required that leverage the burden of their application by reusing existing design and process information and by enforcing the reusability of analyses results allowing early identification of design's weak points and check of design alternatives. This report elaborates on a method that assumes a system specification in an industrial standard notation and allows to perform several formal safety analyses. Based on a collection of failure models and means of specifying safety requirements, the techniques produce results along the lines of traditional methods.

We show how to combine traditional techniques, required by the Aerospace Recommended Practice (SAE-ARP) standards, like Fault Tree Analysis, Failure Mode and Effect Analysis and Common Cause Analysis and also how to automate most of the analysis activities.

The methods described in this paper can be used as means to support the Certification process.

1 Introduction

As avionic systems are getting more complex, it becomes increasingly difficult to perform safety assessment activities required by Aerospace Recommended Practice standards. In particular, the need of having considered all safety-related aspects on a high level of confidence demands for a more systematic and automatic way for performing safety analyses. To address these problems, we present an integrated model-based safety assessment framework, which automates ARP 4761 safety assessment techniques such as Fault Tree Analysis (FTA), Failure Modes and Effects Analysis (FMEA) and Common Cause Analysis (CCA). The

^{*} This work was supported by the European Commission within the projects ESACS (Enhanced Safety Assessment for Complex Systems, FP5), <http://www.esacs.org/>, and ISAAC (Improvement of Safety Activities on Aeronautical Complex systems, FP6), <http://www.isaac-fp6.org/>

framework uses a sound formal approach by employing formal methods to assess the safety of system models implemented in industrial modelling tools such as Statemate [1] or Simulink [2]. We will show the approach to be more comprehensive in respect to the traditional approach since it allows to consider the full temporal properties of the system, for example allowing safety-critical events to be specified by temporal patterns stating that events need to occur in a particular order to be safety-relevant. The underlying analysis techniques elaborate on these specifications and allow, for instance, to identify and relate failure causes and consequences even if they happen at different time instances with several system interactions between them. The possibility to automatically generate results which are easily update-able when the system model changes allows to quickly verify possible design alternatives. This approach can be applied since the first system development phases, therefore identifying potential weak design points with timeliness avoiding later and costly design changes. By applying some of the developed techniques to a given case study we will moreover show that we achieve results beyond those that can be achieved by classical verification techniques [3,4] like model-checking, yet having the same degree of accuracy. The two main enablers for these are, first, a failure injection (c.f. section 3.2) that essentially allows to maintain nominal and dysfunctional system behaviour within one model and, second, the corresponding analysis algorithms (c.f. sections 3.3–3.6). The latter allow to reduce the number of verification runs by a factor that (e.g. in the case of FTA) is exponential in the number of failures.

Typical questions that will be answered by the techniques presented are “Is it possible to violate a certain safety requirement?”, “Which failures and failure combinations need to occur to violate the safety requirement and which additional timing constraints are necessary?”, “Is it possible for a fault to occur undetected?”, “Will a list of impacted items violate independence assumptions underlying a system design?”, “Is it possible to continue a flight in a failed configuration?”, “Will an erroneous flight procedure lead to a safety requirement violation?”. In the following it is shown how such questions can be answered by performing several analyses on the braking system of an aircraft. All analyses were carried out with STSA, a Statemate [1] Plug-in for supporting safety assessment actions. The paper is structured as follows: Section 2 explains the model of a braking system to be used as an example throughout the following sections. Section 3 shows how to apply several traditional and new safety analysis methods on the braking system model. Section 4 discusses related work and section 5 concludes with the indication of future work.

2 Braking System Example

We illustrate our approach by applying it to the model of the wheel brake system as described in appendix L of ARP 4761 [5]. The example was chosen due to the fact that the systems description is relatively concise while retaining a lot of interesting features w.r.t. to safety analysis. Since it is featured in the ARP,

it should be well known to safety engineers. Incidentally, [4] uses the same example (albeit modelled in Simulink rather than Statemate), which gives us the opportunity to directly compare their results with ours. A detailed description of the Statemate model can be found in [6].

The braking system, whose architecture is depicted in figure 1, controls the amount of hydraulic power applied to the brake pistons installed at the main landing gears. Hydraulic power is supplied via two independent hydraulic lines, the normal and the alternate line. The normal line is powered by the green hydraulic pump, the alternate line by the blue one. Additionally, the blue pump is backed up by an accumulator. Thus, the system distinguishes three operational modes, normal (powered by the green pump), alternate (blue) and emergency (accumulator). In normal mode, the valves regulating the flow of the hydraulic pressure are completely controlled by the BSCU (Braking System Control Unit), a computer which computes braking and anti-skid commands. In alternate mode, the BSCU provides anti-skid commands, as long as it is working. The normal braking commands are directly taken from the mechanical position of the brake pedals. In emergency mode, anti-skid is disabled completely.

The BSCU itself consists of two redundant command units and two monitoring units which supervise their respective command units. A switch is used to select the active command unit based on the output of the first monitor. When the monitor signals a failure in the first command unit, the switch selects the second one.¹ When both monitors signal failure of their respective command units, the shutoff signal is issued.

The shutoff signal closes the “Shut Off Selector Valve”, which inhibits the green pressure from reaching the “Selector Valve”. The Selector Valve engages the alternate mode as soon as the green pressure is not available (which can be caused by the described BSCU shutoff or the failure of the green pump). Also, switching between modes can be commanded manually by the pilot.

The presented Statemate model of the braking system is largely a direct formalisation of the model described in the ARP [5]. The main differences are the missing autobrake mode in our model and the restriction to one braking pedal. However, these differences are irrelevant w.r.t. the analyses carried out in the later sections.

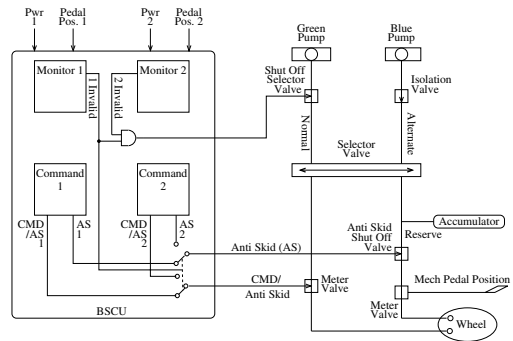


Fig. 1. Architecture of the wheel braking system

¹ Note that our model uses an invalid signal, while the original ARP model uses a valid signal. However, this is equivalent on the model level.

3 A Stepwise Approach to Safety Analysis

Before we start looking at possible failures, causes, and impacts, we assume that the model is free of design errors. The validation of this assumption will be done with available verification tools [3]. For the braking system this is outlined in section 3.1. Having passed this check we extend the nominal behaviour by failure behaviour as shown in section 3.2. After this we pass to more traditional methods used in safety assessment. All of them are applied on the extended model ensuring consistent results. In section 3.3 we are looking for a complete set of causes for a given failure by performing a fault tree analysis (FTA) and a refinement of this analysis allowing the simulation of identified cut-sets. Looking in the other direction, i.e. by investigating failure impacts, we will perform a Failure Mode and Effect Analysis (FMEA) in section 3.4. After this we “jump” over system boundaries and look at causes that arise external to the system. In particular, we will investigate, how to identify *common causes* that lead to a violation of failure independencies. In section 3.5 we show how to perform the necessary Common Cause Analysis (CCA) and how to incorporate the results in the FTA of section 3.3. The final aspects we will consider are “dynamic safety requirements”, i.e. requirements that change over the time of the system’s operation. Such requirements occur in Mission and Reliability Analyses (MRA) and will be investigated in section 3.6.

3.1 Nominal Correctness

The goal in the model-based safety-analysis is to show that all safety requirements hold in the nominal case and the probability for all scenarios containing failures that violate it is within acceptable limits. As an example throughout this text we will use the following safety requirement from ARP 4761:

Loss of all wheel braking during landing or Rejected Take Off (RTO) shall be less than $5 \cdot 10^{-7}$ per flight.

Since we are performing a qualitative rather than a quantitative analysis we do not compute the probability itself. Rather we determine the failure combinations qualitatively that contribute to this probability. So we rephrase the requirement and ask for:

Loss of all wheel braking shall not occur.

Having a formalisation of the model (from section 2), we need to formalise the safety requirement as well. In simple setting such formalisation may be done by characterising a safety-critical state (e.g. by specifying that the pressure of the blue line is above a given limit). If timing is important for the specification of the safety requirement, temporal logic may be used [7]. Although being quite expressive, temporal formulas are sometimes not easy to understand. A reasonable approach is to use *patterns* for typical temporal relationships occurring in safety requirements. For establishing the nominal correctness of the design we use the *ModelCertifier* from OSC [8] that comes with a library of patterns for

typical safety requirements. The most obvious way to formalise the above safety requirement is to state that whenever the pilot pushes the braking pedals, either the green or the blue line have to supply pressure to the wheels subsequently. Due to the stepwise propagation of the braking commands we cannot simply use a safety requirement of the form

(Pedal pressed) implies ((Green pressure high) or (Blue pressure high)).

Instead, we use the following pattern supplied by the OSC pattern library:

`P_stable_X_steps_implies_finally_Q_B.`

This pattern states that whenever P (the pressed pedal in our case) is true for X steps then Q has to hold after at most B steps. In our case, Q is instantiated with the disjunction from above, while we set X and B to 10 milliseconds, each. We require the pedals to be pressed for at least 10 milliseconds in order to avoid situations where the pedal is pressed and released over and over again. Otherwise, we run into situations, where one of the different valves along each line is always closed thus blocking the pressure from reaching the wheels. Another solution to overcome this problem is to change the system to open all the valves for a certain amount of time, whenever the pedals are pressed (or to check whether physical laws ensure this). Using this pattern, we can prove the nominal correctness of the system model with the ModelCertifier. Further verification tasks solvable with the ModelCertifier include checks for non-determinism, races, range violations and others.

3.2 Failure Injection

After the nominal correctness has been established, the next steps will evaluate the system behaviour in case of failures. Thus, the model has to be extended to reflect the possible behaviour of the system in the presence of failures. This is accomplished by injecting a number of (candidate) failures, i.e. failures that *might* occur. As a consequence, the resulting model may behave as in the nominal case or it may behave as if some combination of the injected failures have occurred. In the latter case it is free to apply the failures in any order with any kind of time passing between their occurrences. Figure 2 indicates three possible runs of a system with three failures (F_1, F_2, F_3) injected. It is this kind of generality that allows to do an exhaustive investigation of possible failure scenarios (as outlined, for instance, in section 3.3).

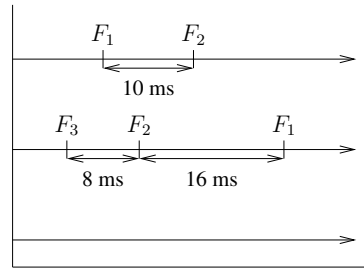


Fig. 2. Different system runs with three injected failures

For the sake of the analyses presented in the following sections we require the failures to be observable, so in addition to the actual failure behaviour an

observer is introduced for each injected failure that is able to detect whether (and when) the failure occurs. By this means we guarantee that nominal and failure behaviour are strictly separated. In particular it allows to check whether failures or failure combinations are *causal* for some safety-critical event. This fact will be exploited in section 3.3 to guarantee *minimality* of cut-sets.

The internal complexity of the failure injection is hidden to the user of STSA, who is only required to identify the affected system variable, selecting a failure-mode that should be applied to the variable, and, if necessary, supply failure-mode parameters (see figure 3). Again, a pattern-based approach is used to provide a list of failure-mode templates including *stuck-at*, *delay*, *noise*, and *random* failures. When the provided failure-mode patterns are insufficient, e.g. if the failure affects several model variables in a complex fashion, or the failure makes use of some internal system mode (degraded system mode), it is possible to apply *user-defined* failures, i.e. failures that are defined within Statemate. In this case the system ensures that nominal and failure behaviour is separated by requiring the user to specify an input that triggers the failure behaviour. After all failure candidates are specified, STSA injects them without any additional interaction in the aforementioned way.

The failure injection was applied to the braking system in the following way: Failures of the electrical and hydraulic power supplies are represented by *user-defined* failure-modes, since the supplies are modelled as inputs to the model. A *user-defined* failure-mode may also be applied to the manual mode selection thus modelling wrong behaviour of the pilot. Failures occurring inside the braking system itself are modelled using *stuck-at* failure modes.

We defined *stuck-at* failure-modes for a variety of variables. Failures of each of the valves are modelled using a *stuck-at* failure-mode with both states (*open* and *close*) as possible values. This way, the failure-mode can force the valve to be stuck in the open position, passing the incoming hydraulic pressure, or stuck close, blocking the pressure. Similarly, the validity signals from the BSCU monitors can either be made stuck *true* or *false*. Further failure-modes were created for the braking commands computed by the two redundant BSCU units and the reference commands the monitoring units compute. Another failure-mode was created to model the failure of the switch unit inside the BSCU.

3.3 Fault-Tree Analysis

After the nominal correctness has been analysed and the system model has been extended with failure-modes it is now possible to perform a fault-tree analysis (FTA). In order to do this the next step is the definition of a *Top-Level Event* (TLE). A top-level event describes a system state that should not be reachable in

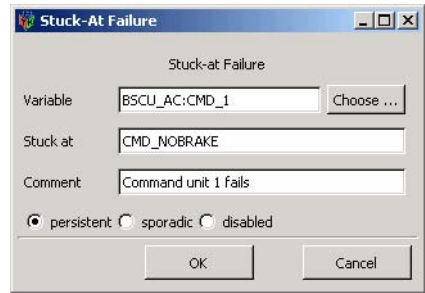


Fig. 3. Specifying a stuck-at failure

the nominal behaviour. In most case there is a direct relation between a top-level event and a safety requirement in the sense that $TLE = \neg SafetyRequirement$ so that is is possible to reuse the definition of safety requirements as they are described in section 3.1.

After all the basic parts of the analysis definition are in place we can start with the analysis. The algorithms we are using to perform an FTA are build on BDD techniques and are related to BDD model-checking. Although our approach shares the fundamental attributes of these techniques it has been extensively enhanced. The main enhancement is in the results that are obtained from the analysis, traditionally model-checking algorithms only analyse whether a certain property can be proven correct or try to find a counter example if the property has been found to be be invalid. In our analysis we are not only interested in a single counter example but in finding all the causes for a TLE. During our FTA analysis a set of all failure-combinations that are necessary to reach the TLE are computed. As these combinations are minimal by construction, they are similar to the well known *minimal cut-sets* that are often used in traditional safety-analysis [9].

Minimal cut-sets are an abstraction notion of a failure-situation. Even for a designer who has an in-depth knowledge of the system it is often not easy to understand the interaction of failure-modes and how they lead to the TLE. In order to ease this task it is possible to use STSA to generate a concrete counter example that shows not only the development of the failure-modes but also that of the the system variables. While the fault-tree generation employs a *free-ordering semantics*, i.e. the order of failures is not restricted in any way, the generation of a counter example produces one concrete ordering. The usefulness of this approach is that the generated fault-tree is complete since it considers *all* possible orderings of failures, while the generated counter examples show a concrete ordering enabling the designer to easily find the deficiencies in the system. One should also note that this kind of analysis is hardly to be achieved by a simulation-based approach, since then one is required to actually specify the order and duration of failures during simulation. Thus to achieve the same quality of result one had to do a lot of iterations.

We applied the fault-tree generation to the braking system using the safety requirement described in section 3.1 (loss of wheel braking) and a subset of the failure-modes described in section 3.2. The INVALID signals of the BSCU monitors and the complete shutoff of the BSCU were used as intermediate events. This results in the fault-tree shown in figure 4. We will have a closer look at the two highlighted subtrees.

Subtree b) is directly related to the one depicted in the ARP [5, page 200], except that we did not include failure-modes for the electrical power or the failure of the hydraulic system (except for the pumps). The subtree shows that the safety requirement is violated, when all three modes fail to operate. The reason for the failure of the normal mode is further broken down to be caused by the failure of the green hydraulic pump or failure of the BSCU units to command braking. Note that this also occurs, when the BSCU monitors fail to

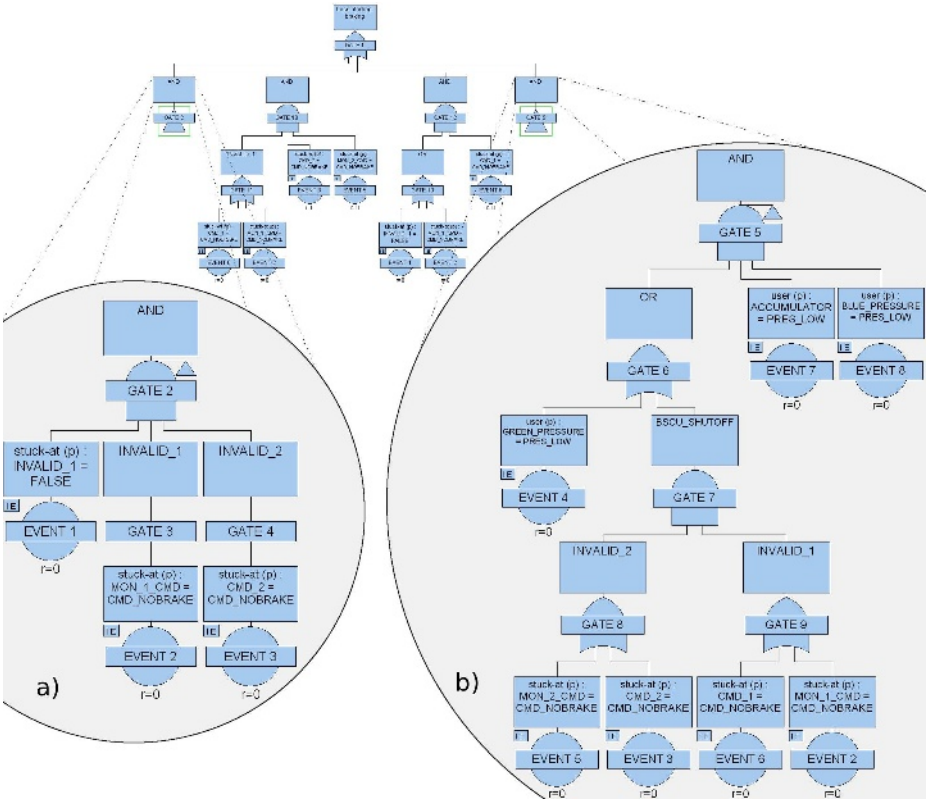


Fig. 4. Auto generated fault-tree

correctly compute the reference signal, which is covered by the events 5 and 2 in the fault-tree.

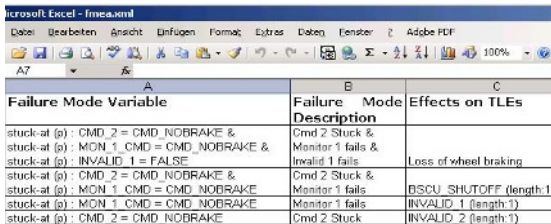
The consideration of monitoring failures is also the reason for the generated fault-tree to be more comprehensive than the one depicted in the ARP, where the occurrence of monitoring failures seems not to be considered. Subtree a) of figure 4 is one of the additional subtrees stemming from this fact. The first remarkable observation of this subtree is the circumstance that it only features failures occurring inside the BSCU, i.e. the TLE is reachable without any failure of the alternate and emergency mode. This is due to the fact that the failure of the monitoring unit inside the BSCU can inhibit its shutoff, thus preventing the system from changing to alternate mode. Nevertheless, a closer look at the minimal cut-set of subtree a) in figure 4 reveals a situation which is confusing at first. The failure of the first monitor (Event 2) activates the first invalid signal, yet the subtree also contains a failure which makes the same signal stay inactive. We can find an explanation for this by letting STSA compute a counter example, which shows the temporal ordering of the involved failures.

Such a trace is depicted in figure 5 which easily explains the odd situation in the fault-tree. One can see, that the first failure occurring is the failure of the first monitor. Due to this, the BSCU switches to the second command unit, which subsequently fails. At last, the invalid signal fails, inhibiting the shutoff of BSCU.

We conclude by referring to [10], where the technique has been applied to the high-lift system of an aircraft.

3.4 FMEA

Each of the cut-sets that has been computed may give rise to other questions, for instance, whether there are other impacts. Since all kind of analyses are performed on unique system and failure models we may proceed to investigate cut-sets for further possible impacts. The traditional way for answering this question is to perform a Failure Mode and Effect Analysis (FMEA), also embedded in STSA. An important difference to the traditional (application of the) FMEA method is that it is possible not only to investigate single failures for their effects, but also failure combinations. In particular the minimal cut-sets that have been identified during fault tree analysis may be imported and further investigated. This idea of having formal tools *guiding* the safety assessment process will be further followed in the following section about the identification of common causes. It is in contrast to [4] where the idea is “to try to pose the right verification questions to formal tools”. Although we agree that standard verification tools can be used *in principle* to perform safety assessment activities (and in fact we use several classical verification engines), they need to be adopted for safety assessment purposes (e.g. the setting in [4] does not even allow to detect causality of failures).



Failure Mode Variable	Failure Mode Description	Effects on TLEs
stuck-at (p) : CMD_2 = CMD_NOBRAKE & stuck-at (p) : MON_1 CMD = CMD_NOBRAKE & stuck-at (p) : INVALID_1 = FALSE	Cmd 2 Stuck & Monitor 1 fails & Invalid 1 fails	Loss of wheel braking
stuck-at (p) : CMD_2 = CMD_NOBRAKE & stuck-at (p) : MON_1 CMD = CMD_NOBRAKE	Cmd 2 Stuck & Monitor 1 fails	BSCU_SHUTOFF (length:1)
stuck-at (p) : MON_1 CMD = CMD_NOBRAKE	Monitor 1 fails	INVALID_1 (length:1)
stuck-at (p) : CMD_2 = CMD_NOBRAKE	Cmd 2 Stuck	INVALID_2 (length:1)

Fig. 6. Auto generated FMEA table

input to the analysis. The other rows depict the effects that different subsets of the failure-modes have on the system. Subsets without any effect on the TLEs are not shown (e.g. the sole failure of the INVALID_1 signal in our example).

Depending on the size of the state space of the model and the length of propagation paths several kinds of analysis techniques are used in the implementation.

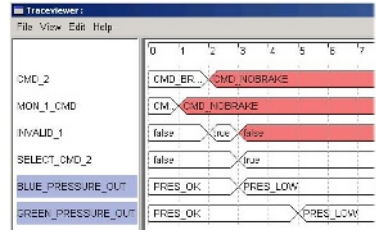


Fig. 5. Traceviewer showing the temporal ordering of failures

Usually FMEA result are given in the form of tables, as shown in figure 6. Each row shows the effect of (a subset) of the failure-modes under consideration, if any. In this case, the first row corresponds to the cut-set depicted in subtree a) of figure 4, which was used as an

Both, SAT-based and BDD-based [11,12] techniques are used in several variants. Again, it is possible to produce a simulation trace for each effect detected during FMEA, thus allowing to trace problems down to concrete runs of the system (model).

3.5 Common Causes

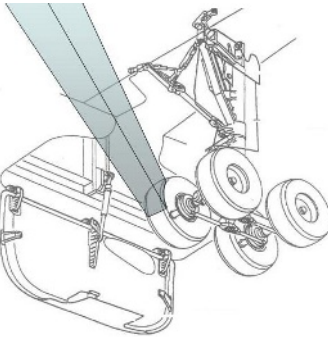


Fig. 7. Tyre burst trajectories

The idea behind failure-modes that have been described in section 3.2 is that all failure-modes are independent. The goal in the common cause analysis is to evaluate the consequences of violations to these independence assumptions. For instance, the violation of independence hypothesis for AND-ed failure events implies the invalidation of basic redundancies foreseen by the aircraft system’s architecture and then different qualitative and quantitative assessment about the violation of safety requirements. As recommended by the ARP, it is therefore necessary to ensure that such independence exists, or that the risk associated with dependence is deemed acceptable in respect to the applicable certification standard requirements.

Typically the violation of independence assumption is caused by an external event establishing dependencies that are not represented in the system model. An example for such an external event could be a tyre burst where the ejected particles hit several components that are independent in the functional design model but closely located within the physical structure (see figure 7).

Taking such dependencies into account requires the integration of the functional models with the the geometrical representation of the system installation. This integration is performed in the context of a stepwise process that starts from the geometry and moves towards the functional tools. The results of safety analyses are then translated back into the geometrical domain to highlight the potential weak points of design installation in terms of violation of safety requirements.

The main steps of the process are detailed in the following. Geometrical models of fragments and relevant trajectories — picked up from a library and built in agreement with the reference standards — are located on geometrical item/s

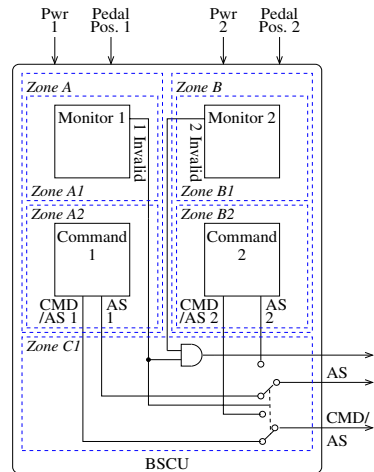


Fig. 8. Zones for the BSCU

potentially interested by a particular risk event (for instance a rotor stage of an engine or a tyre). In this extended geometrical environment (including aircraft systems and fragments trajectories), the consequences of a particular risk event are evaluated by means of interference analysis facilities. The output of this first phase of the process is a list of impacted geometrical components for each possible trajectory of the ejected debris. In the second step of the process the list of impacted items for each trajectory is translated into the list of functional failure-modes through a dictionary. The output of this step is a list of groups of failures that are triggered by the same cause. We call each group of failure-modes a *failure-set*. Each failure-set is identified by the angular coordinates of one or more possible trajectories of the considered fragment. The BSCU from the braking system is divided into several zones (see figure 8). In the example analysis we are presenting there is one trajectory that hits the zones A1 and A2 thereby failing both *Command1* and *Monitor1*. This information is then imported into the analysis tool as a new failure-set that references the failure-modes that have been defined for the affected components.

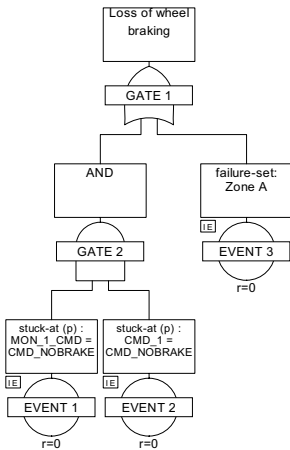


Fig. 9. Fault-tree with Common Causes

As final note, we highlight an alternative use of the failure-set facility. The user has the possibility to bypass the interference analysis by manually creating groups of dependent failures. This could be helpful in order to perform user-guided grouping of failures sharing one or more geometrical attribute. A possible useful application of this manual definition of failure-sets could be in performing the Zonal Hazard Analysis [5]: in this case the failure-sets will be relevant to failures of components located in the same Aircraft zone (see figure 8).

3.6 Mission and Reliability Analysis

So far, our example safety requirement (c.f. section 3.1) did not take different flight phases into account. This section shows, how this can be done by specifying

The last step of forward flow of the process consists in performing the same safety analyses described in previous sections 3.3 (FTA) and 3.4 (FMEA), this time taking into account common causes. To do that, the failure-sets imported into STSA are treated as additional failure-modes in the elaboration of safety results. As we can see from figure 9 the fault-tree now contains a single failure-mode cut-set that is relevant to tyre burst fragment trajectories. This has an obvious impact from the quantitative point of view. The critical trajectories identified in the analysis (i.e. those trajectories that invalidate safety requirements) can be highlighted in the 3D tools afterwards. If the quantitative requirements are not satisfied, our integrated approach eases the verification of possible design alternatives.

As final note, we highlight an alternative use of the failure-set facility. The user has the possibility to bypass the interference analysis by manually creating groups of dependent failures. This could be helpful in order to perform user-guided grouping of failures sharing one or more geometrical attribute.

the successful completion of the aircraft’s mission as the safety requirement. The goal of the mission analysis therefore is to determine to what extent the success of a mission depends on the availability of certain functions (in certain phases of the mission).

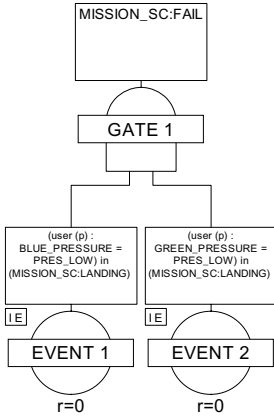


Fig. 10. Fault-tree with mission phases

In order to reason about the aircraft’s mission, a formal model of it is needed. We refer to such a model as the *mission manager*. Usually, it consists of a statechart describing the succession of the different mission phases. Transitions between the phases are annotated with the requirements for the successful completion of the phases. Failure to meet these requirements can either result in the transition to an alternative successor phase or the failure of the mission.²

For the braking system example, we created a mission whose objective is to land on an icy runway. The mission consists of the usual phases of a flight such as *Cruise*, *Descent*, *Landing* and *Taxi*. Assume that the successful completion of such a mission requires the anti-skid function to be working. The mission manager is designed in such a way that upon failure of the anti-skid function, the mission changes the objective to use another runway, which does not require anti-skid. However, this is only possible before the mission reaches the *Landing* phase.

By defining the phases of the mission to be interesting *observables* the analysis can be used to determine the impact of a function loss during a certain mission phase (on the success of the mission). The observables are then used to re-organise the tree that results from the analysis, where the mission phases are reported in the description of the basic events to indicate that the failure-mode of the system items has to occur in a specific phase to have an effect on the missions failure. Figure 10 shows a fault-tree where the occurrence of the failures is only relevant during the *Landing* phase – earlier occurrences do not lead to the failure of the mission (because an alternate runway can be used).

By defining the phases of the mission to be interesting *observables* the analysis can be used to determine the impact of a function loss during a certain mission phase (on the success of the mission). The observables are then used to re-organise the tree that results from the analysis, where the mission phases are reported in the description of the basic events to indicate that the failure-mode of the system items has to occur in a specific phase to have an effect on the missions failure. Figure 10 shows a fault-tree where the occurrence of the failures is only relevant during the *Landing* phase – earlier occurrences do not lead to the failure of the mission (because an alternate runway can be used).

4 Related Work

Model-based verification of fault-trees is shown, for instance in [13,14]. They allow for a very detailed specification of (intermediate) events. The fault-tree itself has to be created manually. However, some work exists how to complete an incomplete fault-tree [15]. The Hip-HOPS [16,17] method is based on the construction of a failure propagation model that is afterwards analysed. The result of the first step is usually a model that is simple in structure when compared with a typical functional model that is created by tools like Statemate [1]

² Where the mission’s failure is considered to be catastrophic.

or Simulink [2]. Another model-based safety analysis approach, derived from methods we jointly developed in [18] within the ESACS consortium, is presented in [4]. The method applies standard verification tools to produce single failure scenarios and is based on a taxonomy of failure models and failure semantics that were developed in [18]. From the safety point of view, however, it suffers from an important drawback: Because standard verification algorithms are used, the method is bound to investigate only single failure scenarios, i.e. each failure combination has to be specified and investigated independently implying that for n failure candidates in the worst case 2^n analysis runs have to be performed to completely traverse the search space induced by the failure combinations. Even this number increases if system modes or mission phases are taken into account. Another weakness of the approach is that it does not take *causality* of failures into account leaving the safety engineer alone with the question of whether a given cut set is *minimal*. In contrast to the pattern-based failure injection presented here, they have chosen a manual injection method that, beside being time-consuming, leaves the system engineer alone with the question of what is the system and what are the failures. It seems questionable whether such manual injection can be carried out practically on larger designs.

5 Conclusion, Further Work

We have presented a tool-supported method for performing a model-based safety assessment. A unique model of the system extended with failure modes — obtained by instantiating pattern from a failure mode library — was used to perform several traditional safety analysis techniques, namely FTA, FMEA, CCA, and MRA. By identifying *causal failure combination* in the ARP 6741 braking system model it was demonstrated that the results are beyond those that are achievable by standard model checking techniques. Other improvements were achieved by exploiting synergies from a combination of analysis results, so that common causes could be incorporated in a fault tree and the role of mission phases could be identified when investigating requirement violation in a MRA. The failure injection and the analysis methods were implemented as a plug-in for Statemate, allowing easy definition and execution of safety analysis tasks. The techniques allows to analyse system models up to 10^{100} states without modifications. Further work includes the adoption of abstraction techniques to address more complex models.

There are no general obstacles in extending the presented analyses to the quantitative, i.e. probabilistic case. The necessary models are at hand (c.f. [19]) and also existing stochastic analysis techniques (e.g. [20]) are waiting to be extended much like we have extended traditional verification techniques to cope with qualitative questions of safety. The challenge will be to apply these to models of realistic size. The necessary state aggregation techniques have already been set up in [21]. Thus, most of the presented techniques are likely to be lifted to the quantitative case in the near future.

References

1. Harel, D., Politi, M.: *Modelling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill (1998)
2. *The MathWorks: Simulink — Model-Based and System-Based Design*. (2004)
3. Bienmüller, T., Damm, W., Wittke, H.: *The STATEMATE Verification Environment - Making It Real*. In: *Proc. of CAV*. Volume 1855 of LNCS. (2000) 561–567
4. Joshi, A., Heimdahl, M.P.: *Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier*. In: *SAFECOMP*. Volume 3688 of LNCS., Springer-Verlag (2005) 122–135
5. ARP4761: *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. Aerospace Recommended Practice, Society of Automotive Engineers, Detroit, USA (1996)
6. Peikenkamp, T., Cavallo, A., Valacca, L., Böde, E., Pretzer, M., Hahn, E.M.: <http://seshome.informatik.uni-oldenburg.de/~sesdocs/safecomp2006> (2006)
7. Clarke, E., Grumberg, O., Peled, D.: *Model Checking*. MIT Press (1999)
8. *I-Logix: ModelCertifier User Manual*. I-Logix, Andover, MA (2002/2003)
9. Vesely, W.E., Goldberg, F., Roberts, N.H., Haasl, D.F.: *Fault Tree Handbook*. NUREG-0492. U.S. Nuclear Regulatory Commission, Washington DC (1981)
10. Peikenkamp, T., Böde, E., Brückner, I., Spenke, H., Bretschneider, M., Holberg, H.: *Model-based Safety Analysis of a Flap Control System*. In: *Proc. of INCOSE*, Toulouse (2004)
11. Drechsler, R., Becker, B.: *Binary Decision Diagrams – Theory and Implementation*. Kluwer Academic Publishers (1998)
12. Somenzi, F.: *CUDD: CU Decision Diagram Package Release 2.4.1*. University of Colorado at Boulder (2005)
13. Schellhorn, G., Thums, A., Reif, W.: *Formal fault tree semantics*. In: *IDPT2002: Interated Design and Process Technology*. (2002)
14. Hansen, K.M.: *Linking Safety Analysis to Safety Requirements*. PhD thesis, Institut for Informationsteknologi, DTU Lyngby (1996)
15. Schäfer, A.: *Combining real-time model-checking and fault tree analysis*. In Mandrioli, D., Araki, K., Gnesi, S., eds.: *FM 2003: the 12th International FME Symposium*. LNCS (2003)
16. Papadopoulos, Y., Maruhn, M.: *Model-based synthesis of fault trees from Matlab-Simulink models*. In: *The International Conference on Dependable Systems and Networks (DSN'01)*. (2001)
17. Papadopoulos, Y., McDermid, J.A.: *Hierarchically performed hazard origin and propagation studies*, Springer-Verlag (1999)
18. Bozzano, M., et al.: *ESACS: An integrated methodology for design and safety analysis of complex systems*. ESREL (2003)
19. Hermanns, H.: *Interactive Markov Chains: The Quest for Quantified Quality*. Volume 2428 of LNCS. (2002)
20. Baier, C., et al.: *Efficient computation of time-bounded reachability probabilities in uniform continuous-time markov decision processes*. *Theor. Comput. Sci.* **345**(1) (2005) 2–26
21. Herbstritt, M., Wimmer, R., Peikenkamp, T., Böde, E., Hermanns, H., Adelaide, M., Becker, B.: *Analysis of Large Safety-Critical Systems: A quantitative approach*. Reports of SFB/TR 14 AVACS 8 (2006) ISSN: 1860-9821, <http://www.avacs.org>.