MANAGING COMPLEXITY AND CHANGE!
INCOSE 2004 - 14th Annual International Symposium Proceedings

Model-based Safety Analysis of a Flap Control System

Matthias Bretschneider
Airbus Deutschland GmbH
Kreetslag 10
21129 Hamburg
Germany

Hans-Jürgen Holberg
OSC Embedded Systems AG
Industriestrasse 11
26121 Oldenburg
Germany

Eckard Böde, Ingo Brückner,
Thomas Peikenkamp, and Harriet Spenke
Kuratorium OFFIS e.V.
Escherweg 2
26121 Oldenburg
Germany

Abstract

Fault tree analysis is a widely adopted technique to systematically analyze causes for a given failure of a complex system. Traditionally, a fault tree is constructed top-down based on knowledge about the structure of the system and the interaction of subsystems. With the increasing system complexity and the accompanying introduction of model-based development techniques in the industrial process, a substantial amount of this knowledge is laid down in the system models. The main focus of the presented techniques and tools is to automatically exploit this knowledge by extracting a fault tree suitable for FaulTree+ directly from a given design modeled in Statemate. The resulting fault tree is complete wrt. the specified failure, i.e. the analysis considers every possible causal failure combination which is guaranteed by applying model checking techniques. Using an aircraft Flap control system this paper shows how to smoothly integrate the technique into an existing model-based process.

1. Introduction

Economical development and quality of life depend to an increasing extent on a more and more complex technical infrastructure. This includes for instance all means of transport, process industry, large construction complexes and energy production. Such systems often carry a significant catastrophic potential at the same time as tolerance in society to accidents is decreasing (very rightly). When problems have occurred in such systems, they very often have had an element of difficulty in understanding the system and anticipate the behaviour of it. Mathematical modelling as a countermeasure has been a longstanding tradition in understanding and analysing complex systems. With the entering of these models in the industrial process, a high demand on adequate, if possible automated, analysis techniques came along, leading to a diversity of tools for different classes of models [MathWorks 2003, Harel 1996]. Since much of the system's complexity stems from the interaction of its subsystems, a diversity of model classes on subsystem level poses a

strong problem in understanding the behaviour of the system. Therefore, when analysing the safety of such systems, it is common to use a simplified model, *safety model*, of the system, prominent examples being Markov-chains or fault trees in the analysis of reliability. A characteristic of these models is that they capture the safety-relevant behaviour at the price of accurately modelling the functional behaviour of the system. The quality of this procedure goes hand in hand with the possibility to accurately set up the simplified model. With the increase of system's complexity the need arises to have tool support in the construction of safety models that in particular guarantees that there is no mismatch between the safety model and the model under investigation. This work focuses on establishing a link between these models by presenting techniques and tools to automatically generate a fault tree from a functional (!) model of a system. A *Statemate* [Harel 1996] model of a Flap control system is extended with failure modes and analysed by STSA, the *Statemate Safety Analysis* [Brückner 2003], to produce a fault tree suitable for import into *FaultTree+* [Isograph 2001].

The work was carried out in the EU funded research project ESACS (Enhanced Safety Assessment of Complex Systems). The technical and scientific objectives of ESACS are to define a methodology to improve the safety analysis practice for complex systems development, to set up a shared environment based on tools supporting the methodology, and to validate the methodology through its application to case studies. These case studies were defined and evaluated by industrial partners. The underlying methodology was developed together with the technology providers, who also were responsible for providing the required technology and its integration with existing modeling and safety tools.

The paper is organized as follows: The next section "Case Study: Flap Control" introduces the underlying case study. The following section 3 "Scope of Analysis" describes which aspects of the system were analyzed and what complexity the system model represented from an analytical point of view. The preparatory verification of the correctness of the nominal system model is discussed in section 4 "Verification of system requirements", followed by an introduction to the applied safety analysis techniques in section 5 "Fault Tree Analysis" which also contains detailed examples of the results gained by its application to the case study. In section 6 "Related Work" different approaches to the improvement of model based safety analysis are presented while the closing section 7 "Conclusion" summarizes the lessons learnt and gives an outlook on the future development of the presented technology.

2. Case Study: Flap Control

To put the proposed methodology into an industrial context, a case study was prepared. Subject of investigation is the controller of the flaps of an aircraft. The flaps are part of the high-lift system of an Airbus. It is well known, that the lift-coefficient of a wing depends on its camber. Deflecting the trailing edge downwards can change the wing camber. The hinged trailing edge is known as a Flap. On the Airbus the Flaps can be in five configurations (positions). The pilots can select the configuration of the Flaps manually by means of the Slat/Flap Lever. The Command Sensor Unit (CSU) converts a selection into a set of discrete electrical signals. These signals are dispatched to their respective channels within a Slat-Flap Control Computer (SFCC). (See figure 1). The new commands are validated within each computer. Each computer independently signals the associated hydraulic solenoid valves on the Power Control Unit (PCU) to release the power off brake and to supply the hydraulic motors to drive the Flaps to the new position. When the PCU position and the demanded position are within a specified threshold range ahead of target position, the system will slow down, until the PCU achieves the actual demanded position. At

this point the respective brakes are applied and the PCU is shut down. This concludes the so-called normal drive sequence. The controller has to take into account failures such as low pressure and power interrupts, or turnaround commands. So there is a multitude of possible drive sequences, which contribute to the complexity of the controller.

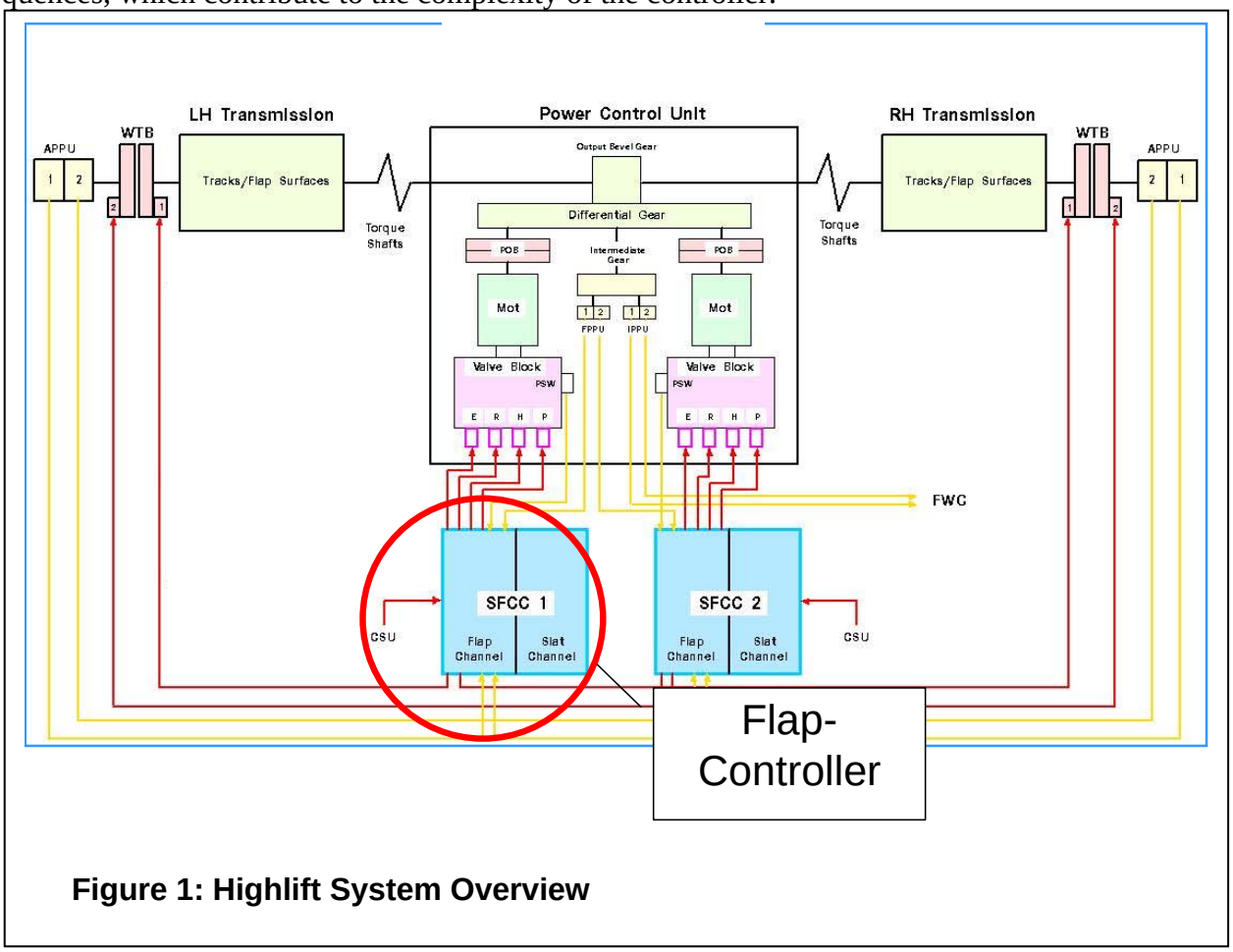

Figure 1: Highlift System Overview

There is also an automatic mode of operation, which will retract the Flaps to relief Flap loads. The Flaps are retracted from the extended position to the next lower lever setting, if the aircraft speed exceeds a certain threshold speed for the selected configuration for more than a specified limit. The selected position will be automatically commanded again if the aircraft speed falls below the threshold speed.

The system is designed in a way that makes critical events extremely improbable. Critical events that may significantly impact aircraft safety include the powered runaway of the Flaps, inadvertent Flap retraction due to auto-command, overspeed and asymmetric extension. System protection against the events is achieved by sensing and monitoring of PCU and Flap positions and rate of change of positions.

3. Scope of Analysis

3.1 Model of Flap Controller

Scope of the analysis is a model of one channel of the Flap-Controller. Its interfaces are shown in the figure below (figure 2). The inputs are the Lever Position (IN_LEV_POS_DI), Air Data and

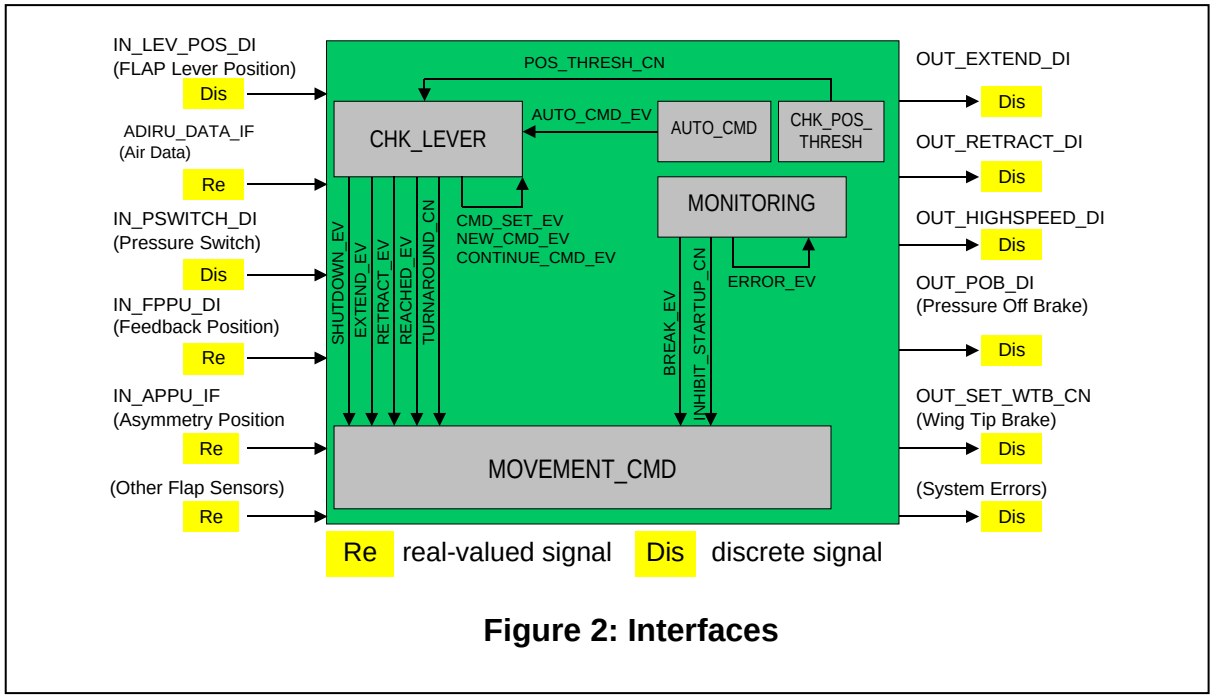

Figure 2: Interfaces

so on. The outputs Extend and Retract control the direction of the hydraulic motor (PCU). The motor has two modes of operation (high speed and low speed). Further the Pressure Off Brake and the Wing Tip Brakes can be used to inhibit movement of the Flaps. The model was built in the Design Office. The purpose is to validate a textual specification, i.e. whether it is technically feasible to implement textual system requirements. The complexity of the Flap Controller results from a multitude of driving sequences: there is the Normal Drive Sequence, there are sequences involving Turnaround, sequences taking into account low hydraulic pressure, power interrupt and so on.

The Flap Controller is a directed system: all interfaces can be classified into independent (inputs) and dependent variables (outputs). (The values of an independent variable strongly influence the complexity of the analysis as indicated in section 3.2; during simulation such a variable would be an input to a simulation.) The controller is made up out of several components working in parallel. The behaviour of a component is described by a finite state machine (FSM). The FSM communicates with other FSMs by means of signals and events. As an example consider the INHIBIT_STARTUP_CN signal that is set after the system has detected failures of subsystems and in this case inhibits startup of the drive sequence (retraction/extension). For this kind of systems the STATEMATE tool turned out to be appropriate (For Statemate see reference [I-logix]).

3.2 Model Size and Complexity

A major reason for applying automatic techniques to safety analysis is the complexity of controllers. We mention some characteristics of the model. Regarding static characteristics the Flap Controller consists of 30 charts ("pages"); most of them are instances of generic chart definitions. The number of data items in the Data Dictionary is 164; in addition 38 Conditions and 12 Events are used. Most of the data items are Floating Point variables and constants defined in complex structures like arrays and records. Complex data types are all defined in User-Defined Type definitions (9) and all periodic calculations are defined as sub-routines. The dynamics of the model involves 7 timers, which count up to 200 steps. Timers have a great impact on the model diameter (the longest path in the state transition system underlying the model). From this is obvious that this model cannot be analyzed by hand.

Even automatic analysis is non-trivial. The translated binary model representation of the Flap Controller consists of 469 state bits and 75 input bits. The flat finite state machine has about 35.000 nodes. To model the behaviour explicitly by finite state machines would require 35000 states. To understand the complexity of the computation one should realize that the result achieved by the analysis technique is equivalent to an "exhaustive test". Exhaustive testing would amount to examining all possible combinations of input values (i.e. 2^{75}) for each step. The number of steps is determined by the diameter of the model.

4. Verification of System Requirements

The first step when doing an automatic safety analysis is to execute a set of simple robustness checks (e.g. to find non-determinism and write/write or read/write races). After that it should be ensured that the nominal model fulfills all safety requirements. When these preliminary steps have been taken care of, the actual fault-tree analysis (FTA) can start. To demonstrate the technique the following property has been selected:

The Flap System outputs RETRACT and EXTEND shall never be true at the same time.

These two outputs of the controller set external hydraulic values that select the direction for the PCU. Setting both extend and retract by the controller is clearly a problem and therefore should not be possible. From this requirement it is possible to formalise the definition of the top-level event (TLE) that can be used during the FTA (figure 3).

```
TLE = OUT_RETRACT_DI and OUT_EXTEND_DI
```

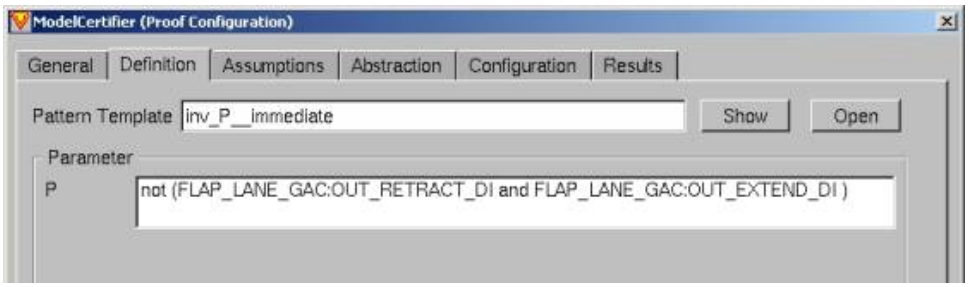

Figure 3: Definition of the TLE

Besides these rather simple properties it is also possible to import more complex safety requirements from the *ModelCertifier* [I-Logix 2003]. The pattern selection itself and the instantiation of pattern parameters is done within this verification tool.

5. Fault Tree Analysis

5.1 Definitions (Fault Tree, Minimal Cut Set)

The Fault Tree Analysis (FTA) is an analytical technique that is used for Safety Analysis. An FTA attempts to integrate all factors that affect a failure of a system into a single FTA Logic Diagram. The symbols used in a single FTA Logic Diagram are called Logic Gates and are similar to the symbols used by electronic circuit designers (see Figure 4). The box on the top of the tree (Top Level Event, TLE) represents the system failure. The leaves x(1), x(2) etc. denote component failures (Basic Events). Formally, an FTA Logic Diagram is a graphical representation of a Boolean function *f: B^N →B*, where B={True, False}; the literals True and False denote Failure and Normal Operation, respectively, and N is the number of components of the system. So, for x in *B^N*, x(i)=True means that component i is failed. The example tree is *f(x(1),x(2),x(3)) = x(1) and (x(2) or x(3))*. Typically, classical FTA can be defined by means of And- and Or-Gates; they do not contain the Negation (Monotonic System, see [Gaede 1977]). Thus the Boolean function *f(x(1),…,x(n))* can be transformed into the so-called Disjunctive Normal Form (DNF): *f = A or B or C …* , where the terms A, B, … are conjunctions of Basic Events. The DNF is uniquely determined by f up to ordering. The terms A, B, are denoted as Minimal Cut Sets (MCS). For the example *A=x(1) and x(2)* is a MCS. The interpretation of A in the context of the Fault Tree is: if A is True, i.e. both components 1 and 2 have failed, the system fails. Note, that due to uniqueness of the DNF, none of the terms B, C, … can be equal to x(1). It follows that when 1 failed and 2 in normal operations the system is normal.

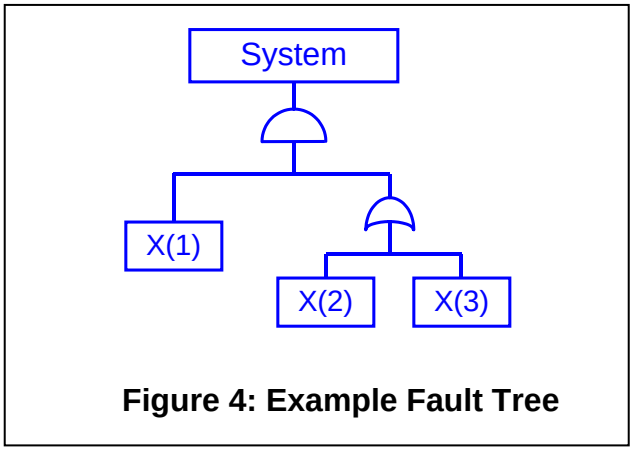

Figure 4: Example Fault Tree

When an FTA analysis is done *manually*, the safety analyst would develop the Fault Tree top down by tracing back the TLE for its possible causes. This analysis is based on engineering judgement and tests.

5.2 Model-based Fault-Tree Generation

In the model-based approach discussed in this paper, the Fault Tree for a given Top Level Event is constructed from MCS. The Fault Tree is the disjunction of MCS. Based on the model (with injected failures) it is determined whether there exist runs (scenarios) involving component failures that lead to the undesired TLE. If a run with this property could be found, the information which component failures occurred during this run is extracted. If, say, components 1 and 2 were involved, a term `A=x(1) and x(2)` would be created.

Actually building all runs of the system is prohibitively expensive. The aforementioned

number of 75 inputs allows to feed at each step of the model execution one of 2^{75} possible inputs into the system, each of them possibly leading to another target state. Therefore, the actual computation makes use of symbolic analysis techniques that are very similar to techniques used in verification of finite state systems. In contrast to conventional methods the main advantage of these verification techniques is that a complete analysis of the state space is achieved without having to simulate every possible system run.

Before the FTA is carried out, failures are injected into the system representing known failure possibilities of components. Instead of explicitly modeling failures, a library of patterns for failure modes is provided that cover the possible failure types: "Stuck-At" failure modes, "Ramp Down" failure modes representing failures leading to an increasing deviation from a nominal value, "Noise" failure modes representing, for instance, noise on an electric connection, "Delay" failure modes representing a delay in the propagation of nominal values. Different failure modes can be combined. It is possible, for instance, to define the delay of some value ("Delay") and, at the same time, have a noise failure present on the line ("Noise").

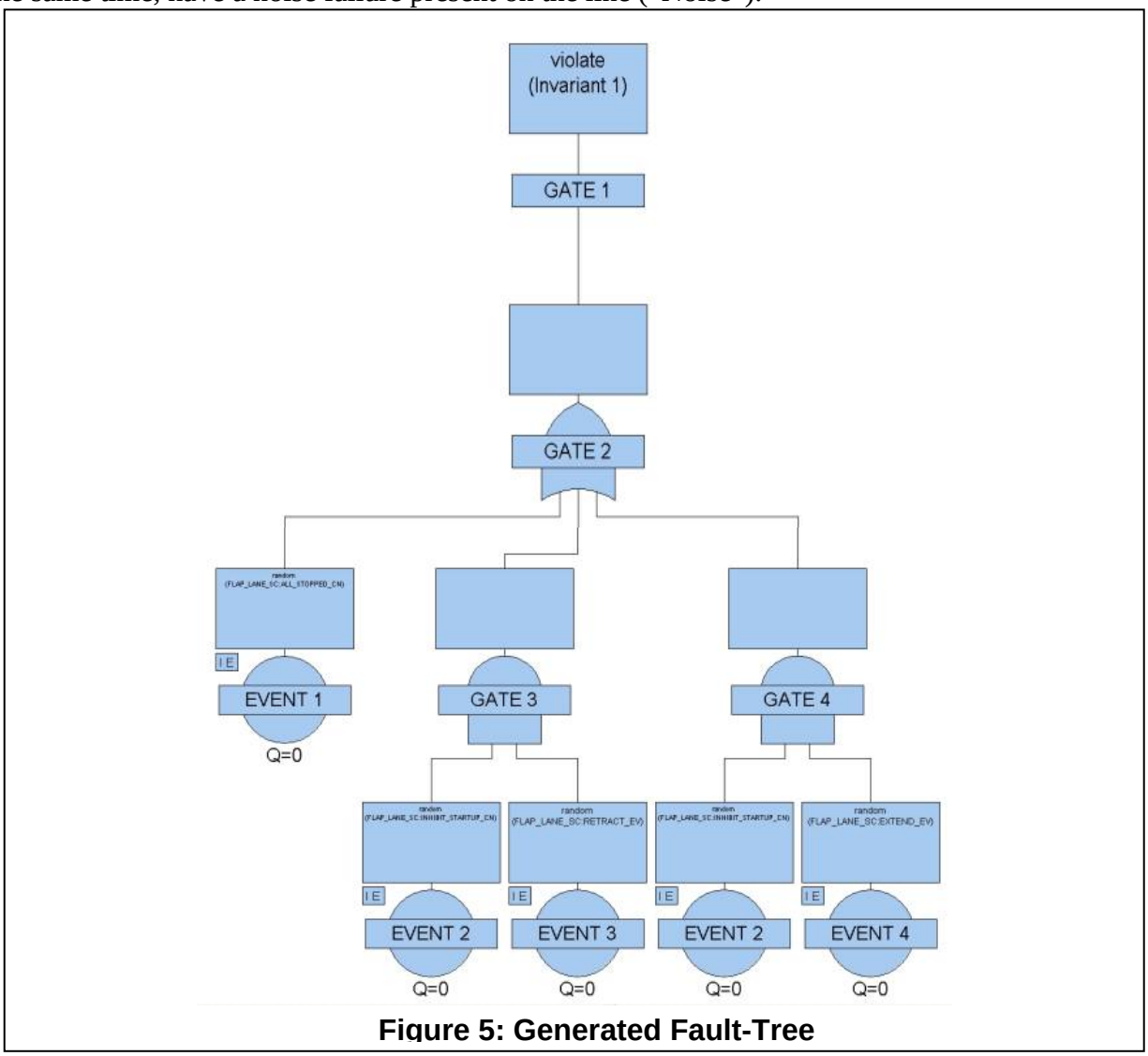

Figure 5: Generated Fault-Tree

Of particular importance in the failure mode library is the "Random" failure mode. It represents a behaviour where at each time step in the model execution a random value is taken for the

variable affected by the failure. The importance of this failure mode stems from the fact that it includes all other failure modes in the following sense: Whenever there is a run of the system exhibiting a certain failure induced by one of the above failure modes, then the same run is possible, if the failure was induced by a random failure mode. As a consequence, random failure modes are the most general failure modes and if the analysis shows that a random failure applied to some component "doesn't harm" (i.e. does not lead to the TLE) then this is true if one replaces this failure mode by one of the more specific failure modes.

Following the above strategy we insert "Random" failures for the signals between the subsystems (see figure 2). By applying the "Random" failure mode to the signal INHIBIT_STARTUP_CN, for instance, we inform the analysis tool that during fault tree computation it should consider all possible failures on that signal (also in conjunction with possible failures on other signals). Intuitively, a failure on that signal means that the monitoring functionality has failed.

5.3 Fault Tree Analysis of the Flap Model

After having defined the TLE and all possible failures of the system, the analysis is started and returns a fault tree suitable for FaultTree+ [Isograph 2001] where also quantitative (probabilistic) analyses can be performed. For the Flap model we obtain the following fault tree (figure 5).

Three failure combinations (cut sets) were computed that are responsible for the violation of the invariant. This fault tree is complete for the given TLE and the injected failures, so we know that there are no other failure combinations that may cause the TLE. On the other hand we see that the second cut set consists of two basic events where "Random" failures occurred on the INHIBIT_STARTUP_CN and the RETRACT_EV signal. If we are satisfied with the result because, say, we know how these two failures interact to produce the TLE, then we may proceed now with a quantitative analysis to ensure that the probability of the TLE is "small enough" for the purpose of the application. If on the other hand, we do not know how the two failures interact to produce the TLE, we might ask the tool for a simulation run for this cut set. As a result we obtain a simulation script (Simulation Control Program, SCP, in Statemate's terminology) producing input stimuli for the design such that the failures of the selected cut set occur *and* the TLE is observable at the end of the produced simulation. For the above two-element cut set we obtain an SCP that, when fed into the simulator, produces the following waveform (figure 6).

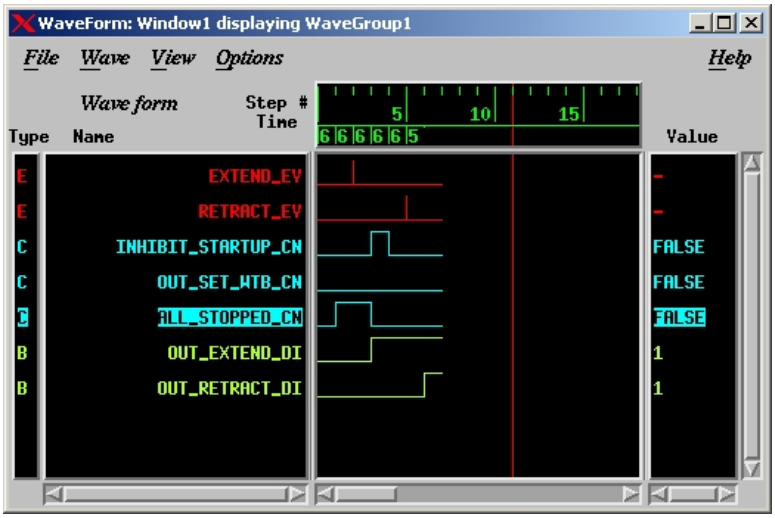

Figure 6: Simulation of an MCS in Statemate

5.4 Algorithmic aspects

The techniques used in the implementation rely on efficiently traversing huge state spaces. Such techniques were introduced for the verification of sequential circuits in [Coudert 89] and [Touati 90]. They rely on ordered binary decision diagrams (OBDDs) to efficiently represent state spaces and transition relations of typical design models. Later these data structures were also used in efficient representation of fault trees [Coudert 93]. The techniques used in the presented approach combine the two ideas by exploring the state space of the model with OBDD techniques and, at the same time, generating the fault tree. This allows to efficiently generate the cut sets of the tree simultaneously.

The generated fault tree is internally represented as an OBDD, guaranteeing that only "relevant" events are stored. Cut sets (or prime implicants for non-monotonic models) can be generated as shown in [Coudert 93]. The hierarchy of the tree is not visible in the OBDD. Currently, the necessary structuring can be done within the FaultTree+ Tool [Isograph 2001]. In the future, a set of *structuring hints* based on the hierarchy and dataflow of the design will be computed.

In parallel to performing the case study, a prototypical version of the tool has been used to evaluate the introduction of abstraction techniques known from the verification of finite state systems. By exploiting the techniques a considerable performance improvement was observed that, in the analysis of complex TLEs, reduced the run-time of the analysis from several days to 2-3 hours. The abstraction technique is safe, i.e. no cut-sets are dropped. But it is possible that some of the cut sets are too small, i.e. too pessimistic. This is not a problem from a safety point of view, but sometimes not satisfactory. In this case the aforementioned possibility to produce an SCP proves useful: it is obvious that for cut-sets that are "too small" no SCP can be generated. This fact can then be used to iterate the analysis with a refined focus for the "too small" cut sets.

6. Related Work

BDD-based fault tree representations have first been investigated in [Coudert 1993]. The techniques presented here are based on that work, but also extract the information necessary to construct the fault tree from a system model.

Another approach to model-based failure analysis is MDS (see [Mauss 2000]): The component models contain explicit description of nominal and faulty behaviour. The behaviour is described using a set of constraints relating the local variables of a component. For instance a valve model may have three behaviour modes 'ok', 'stuckOpen' 'stuckClosed', representing nominal behaviour, and two different component faults. The technique for deriving Fault Trees is based on the Assumption Based Truth Maintenance System Approach. The main data structure is a net, which can be traversed forwards and backwards to produce Failure Mode Effect Analysis and Fault Trees, respectively. MDS performs forward simulation of the component models (concurrent finite-state machines), until a steady state is reached, or a cycle is detected.

Another approach is presented by the SETTA project [Papadopoulos 2002]: The analysis is applied to Simulink Models. The component description is annotated by a data flow failure model. The annotations involve the description of failure propagation and transformation for the component. Failures include omission, commission and value failures. As an example, the user has to identify for a value failure related to an output of a component which causes might affect this output (internal faults, erroneous inputs). Given this input for each component of the model, the fault tree relating to the system is generated by traversing the model backwards starting from unintended output (data flow).

7. Conclusion

In this paper we presented a model-based fault tree analysis based on Statemate designs. It explains the main functionality of the STSA (Statemate Safety Analysis) tool and its application to the verification of a complex industrial application, the Airbus Flap control system. The tool generates fault trees using algorithms based on formal methods techniques working directly on a detailed system model.

The underlying principle for the model checking based fault tree construction is as follows: In the first step the nominal correctness of the system design is verified. After that, failures are injected into the system model and a top level event representing undesired behaviour is defined. There exists a wide range of standard failure modes like random, stuck-at, delay or noise to cover all failure situations. The algorithm starts from this extended system model and the TLE and generates, using formal symbolic model checking techniques, a Boolean formula representing all the possible ways in which the top level event is not satisfied by the extended system model. From such a formula it is possible to extract all the possible combinations of failures of components.

If the design model behaves correctly with respect to the top level event (i.e. if the top level event is verified by the design model), such combinations of failures are exactly the reason for the top level event not being fulfilled anymore in the extended system model, because the failure modes are the only change between the design and the extended system model. Standard minimization techniques are then run on the combination of failures identified, in order to extract the combinations that are minimal. The algorithm outputs are fault trees in a standard format, which show the relevant cut sets causing the failure situations. This data is visualized in a standard tool as a graphical fault tree, which can be used for all further analyses, e.g. failure probability estimations.

It has been shown, that even with a complex model like the Flap system, the automatic STSA tool can be used in an industrial environment. When dealing with the state explosion problem, especially the usage of data abstraction technology is unique in the formal safety analysis context and it makes the whole technology scaleable on industrial sized applications.

Furthermore, it has been proven that the nominal behaviour of the Flap system satisfies the given safety requirements. However, non-trivial fault trees have been found indicating that the robustness of the design for certain failure combinations could be improved. The STSA tool found, for example, a specific trace where two different failures have been injected in a specific ordering, so that the Flap system generates retract and extend commands at the same time.

Two important advantages of this fault tree generation have been identified. First, the generated cut sets are complete (wrt. the given model and failures), i.e. no error combination, which activates the top level event of the fault tree, is missing. Second, only minimal cut sets are generated, that means, the user can focus on the relevant information.

Additionally, the safety tool allows the error trace generation for each found cut set. These traces are converted into a simulation control program. This visualisation is a convenient way to support the developer in the identification of the dynamic behaviour leading to the undesirable (combination of) events. The automation of the analysis enables the simple iteration of the FTA on the improved design and thereby leads to a better integration of design and safety activities.

References

Bozzano, M. et. al., "ESACS: an integrated methodology for design and safety analysis of complex systems", ESREL 2003

Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., and Hwang, L.J., "Symbolic Model Checking: 10^{20} States and Beyond", *Information and Computation*, 98(2), 1992.

Brückner, I., Lüdtke, A., and Peikenkamp, T., "Statemate Safety Analysis (STSA) Manual", OFFIS, Oldenburg 2003

Coudert, O., Berthet, C., and Madre, J.C., "Automatic Verification Methods for Finite State Systems", *Verification of Synchronous Sequential Machines based on Symbolic Execution*, LNCS 407, Springer Verlag ,1989.

Coudert, O. and Madre, J. "Fault Tree Analysis: 10^{20} Prime Implicants and Beyond". In *Proc. Annual Reliability and Maintainability Symposium*, 1993.

Papadopoulos, Y., "Model-based On-line Monitoring Using a State Sensitive Fault Propagation Model", in SAFECOMP 2002, *21st Int. Conf. On Computer Safety Reliability and Security*.

Gaede, K.-W., *Zuverlässigkeit – Mathematische Modelle*, München, Wien,1977.

Harel, David and Politi, Michal, *Modelling reactive systems with Statecharts: The Statemate approach*, Technical Report, i-Logix Inc., 3 Riverside Drive, Andover, MA 01810, June 1996.

I-Logix, "ModelCertifier User Manual", I-Logix, Andover, MA, 2002/2003.

Isograph Ltd. "FaultTree+ for Windows – Version 10.0", Isograph Ltd. 2001

Mauss, J., May, V., Tatar, M., "Towards Model-based Engineering: Failure Analysis with MDS", *ECAI-2000 Workshop on Knowledge-Based Systems for Model-Based Engineering*, Berlin, 22.08.2000.

MathWorks, "MATLAB Simulink/Stateflow", http://www.mathworks.com, The MathWorks 2003.

Ravi, K. and Somenzi, F., "High-Density Reachability Analysis", ICCAD, 1995.

Touati, H., Savoj, H., Lin, B., Brayton, R.K., and Sangiovanni-Vincentelli, A.L., "Implicit Enumeration of Finite State Machines using BDDs", ICCAD, 1990

Biography

Matthias Bretschneider, Dr.rer.nat, Airbus Deutschland GmbH, Senior Expert for Safety and Reliability Methods.

Hans-Jürgen Holberg, OSC GmbH, Director Consulting

Eckard Böde, Dipl.-Inform., Kuratorium OFFIS e.V., Safety Critical Systems Department, Modelling and Verification of Embedded Systems

Ingo Brückner, Dipl.-Inform., Kuratorium OFFIS e.V., Safety Critical Systems Department, Modelling of Safety Critical Systems

Thomas Peikenkamp, Dipl.-Inform., Kuratorium OFFIS e.V., Safety Critical Systems Department, Modelling and Verification of Embedded Systems

Harriet Spenke, Dipl.-Inform., Kuratorium OFFIS e.V., Safety Critical Systems Department, Modelling of Safety Critical Systems