ELSEVIER

Reliability Engineering and System Safety 71 (2001) 229–247

RELIABILITY ENGINEERING & SYSTEM SAFETY

www.elsevier.com/locate/ress

Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure

Y. Papadopoulos[a,*], J. McDermid[a], R. Sasse[b], G. Heiner[b]

[a]*Department of Computer Science, University of York, York YO10 5DD, UK*
[b]*DaimlerChrysler AG, Research and Technology, D-10559 Berlin, Germany*

Abstract

This paper introduces a new method for safety analysis which modifies, automates and integrates a number of classical safety analysis techniques to address some of the problems currently encountered in complex safety assessments. The method enables the analysis of a complex programmable electronic system from the functional level through to low levels of its hardware and software implementation. In the course of the assessment, the method integrates design and safety analysis and harmonises hardware safety analysis with the hazard analysis of software architectures. It also introduces an algorithm for the synthesis of fault trees, which mechanises and simplifies a large and traditionally problematic part of the assessment, the development of fault trees. In this paper, we present the method and discuss its application on a prototypical distributed brake-by-wire system for cars. We argue that the method can help us rationalise and simplify an inherently creative and difficult task and therefore gain a consistent and meaningful picture of how a complex programmable system behaves in conditions of failure.

Keywords: Automated safety analysis; Mechanical fault tree synthesis; Software hazard analysis; Safety cases

1. Introduction

Classical safety analysis techniques such as Functional Failure Analysis (FFA) [1], Hazard and Operability Studies ((HAZOP)) [2], Failure Modes and Effects Analysis (FMEA) [3] and Fault Tree Analysis (FTA) [4] have demonstrated real value over the years and they are still widely practised in safety assessments. Indeed, those safety studies still form the spinal element of the safety case, and provide a frame for the interpretation of the results from other, usually more localised, verification activities such as testing and the application of formal methods. As the complexity of modern programmable electronic systems increases, however, the application of classical techniques is becoming increasingly more problematic.

The first problem that can be observed is inconsistencies in the results from the various safety studies of the system which mainly arise from the selective and fragmented use of different methods at different stages of the design lifecycle. Classical techniques assume different design representations that reflect different levels of abstraction in the system design. While FFA requires only abstract functional descriptions, for example, HAZOP and FMEA require architectural designs of increasing detail and complexity. The problem here lies in that these different design representations are often inconsistent. One of the reasons for that inconsistency is that different notations are employed at different stages of the lifecycle. Perhaps more importantly, abstract designs are not always kept updated, and they do not reflect changes made in lower level designs. Inevitably, the analyses that are based on inconsistent designs are themselves inconsistent. One significant conclusion that in our view can be drawn from this discussion is that if we wish to *address the problem of inconsistencies in the analyses then we must find ways to guarantee the consistency of the design as this evolves in the course of the lifecycle*.

A second problem in classical safety analysis is the difficulty in relating the results of the various safety studies to each other and back to the high-level FFA. One dimension of this problem is that hardware safety analysis and software hazard analysis typically form two separate parts of the assessment and, as a consequence, the relationship between hardware and software failure often remains vague and unresolved. A second dimension of this problem is that, as the analysis remains fragmented, the safety case usually fails to offer a coherent and complete picture of the ways

* Corresponding author.
E-mail addresses: yiannis.papadopoulos@cs.york.ac.uk (Y. Papadopoulos), john.mcdermid@cs.york.ac.uk (J. McDermid), ralph.sasse@daimlerchrysler.com (R. Sasse), guenter.heiner@daimlerchrysler.com (G. Heiner).

0951-8320/01/$ - see front matter
PII: S0951-8320(00)00076-4

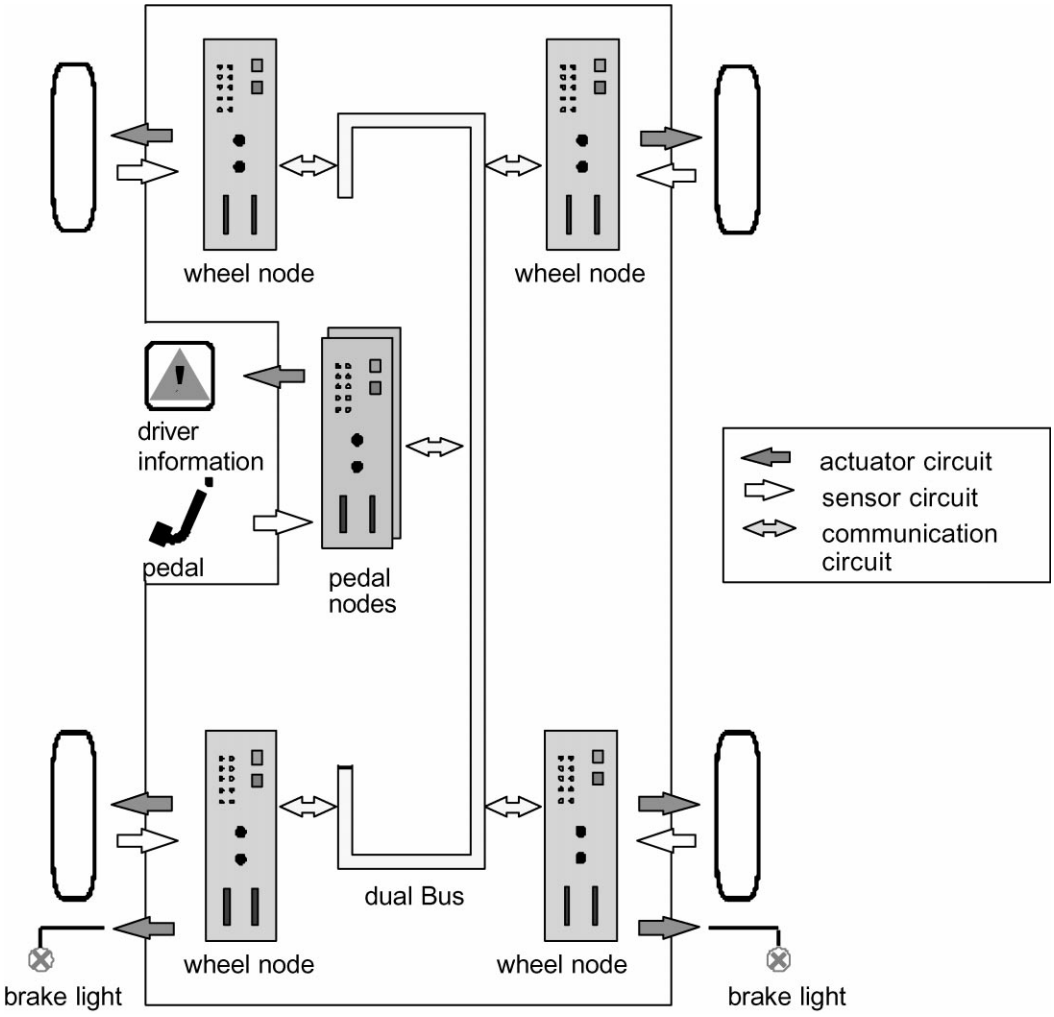


Fig. 1. Architecture of the brake-by-wire system.

in which low-level component failures contribute to hazardous malfunctions of the system. Although fault trees are built for this purpose, the traditional process of constructing these fault trees relies heavily on expert knowledge, and lacks a systematic or structured algorithm which the analyst can apply on a system model in order to derive the tree. In the context of a complex system this process becomes tedious, time consuming and error prone, and the resultant fault trees are large, but more importantly, difficult to interpret and verify. In consequence, safety analyses are in practice not only voluminous but also fragmented and inconsistent. Such analyses are also difficult to interpret and do not always provide a useful resource in the design of the system. But is it not the aim of safety analysis to improve the system design? And does the fragmentation of classical techniques not compromise this aim?

Our first aim in this paper is, precisely, to propose a *new method for safety analysis*, which, we believe, can address some of the difficulties that we discussed above. Our second aim is to demonstrate how the proposed method has helped us analyse, and improve in our case study, the failure behaviour of a prototypical brake-by-wire system for cars, and how at the end of the assessment process we have achieved a consistent and meaningful safety case for this system. The brake-by-wire system not only provides the "case study" in this paper, but it also serves as a "running example" for the presentation of our approach to safety analysis. It is, therefore, useful to start with a brief introduction to this system.

The brake-by-wire system is a prototype in a laboratory environment that has been developed by DaimlerChrysler Research in the context of the European Commission funded project Time Triggered Architectures (TTA[1]) [5]. The system provides a design concept for future brake-by-wire applications in the automotive industry. The general topology of its architecture is illustrated in Fig. 1.

The system is implemented over a network of six

[1] ESPRIT project 23396. The work that we present in this paper continues in the context of a new European Commission funded project called SETTA (System Engineering for Time Triggered Architectures-IST project 10043).

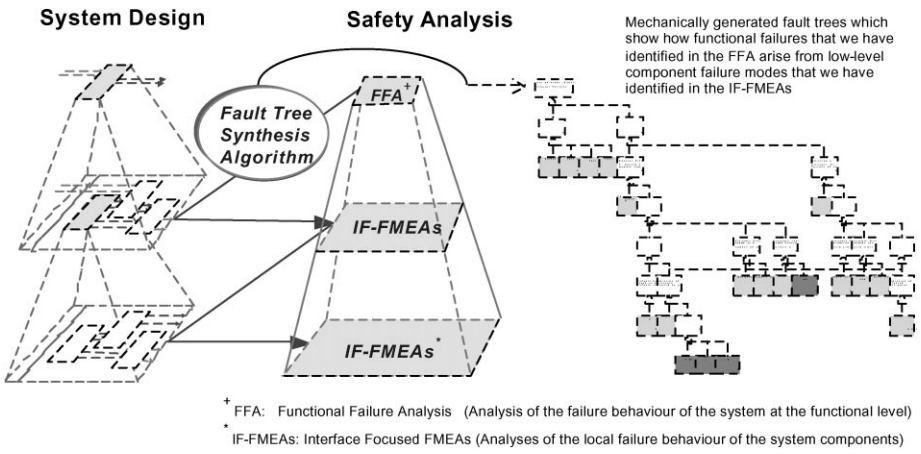


Fig. 2. Overview of design and safety analysis in HiP-HOPS.

programmable electronic nodes which communicate using TTP/C [6], a Time Triggered communication Protocol.[2] Two of those nodes, the pedal nodes, are physically located near the braking pedal. Their function is to read continuously and broadcast the braking demand on two replicated busses. On the receiving end, there are four wheel nodes, which receive the braking demand sent by the pedal nodes. By processing this information and sensory feedback from wheel-load, rotational acceleration and pressure sensors, each node calculates the value of the braking pressure that is fed to an actuator which then applies the actual braking pressure on the corresponding wheel of the car. The overall system delivers a number of sophisticated braking functions, which include braking proportional to each wheel's load, anti-lock braking (ABS) and electronic stability functions. This prototype is linked to a simulation model of the dynamic behaviour of a passenger car on the road. The on-line visualisation component of this simulator can dynamically show the reactions of the vehicle to control commands, injected failures and automatic recovery actions.

In the following sections, we will discuss the way in which we have analysed and improved the behaviour of this system in conditions of failure. Before we do so, though, let us first develop the general method that we propose for the assessment of programmable electronic systems.

[2] TTP/C has a number of safety directed properties which make it particularly suitable for application in safety critical systems. In fault free conditions, for example, the communication controller ensures timely delivery of all communication messages on the basis of a statically defined message schedule. In the presence of faults, the controller *fails silent* in response to transient or permanent faults that could corrupt the temporal access pattern and the integrity of data on the bus. The controller also provides rapid fault detection of certain classes of host failures, bus failures, other node failures and failures caused by disturbances during transmission. Finally, the protocol offers support for replicated buses and replicated nodes and enables the implementation of fault-tolerant architectures [7].

2. Overview of the safety analysis method

The proposed method is called Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) and enables the integrated assessment of a hierarchically described system from the functional level through to the low levels of its hardware and software implementation.

To ensure the transferability of the vast practical experience that classical analyses incorporate, we have founded the new method on a number of well-established techniques such as FFA, FMEA and FTA. At the same time though, we have *modified*, *automated* and *integrated* these techniques to overcome some of the difficulties that we have already discussed. The method mechanises and simplifies a large and traditionally problematic part of the analysis, the development of fault trees. It also integrates classical hardware safety analysis with software hazard analysis and guarantees the consistency of the results from the assessment.

Fig. 2 illustrates the safety analysis process in HiP-HOPS. The process starts early in the design lifecycle with exploratory FFA of an abstract functional model of the system. At this stage, we employ an extension of classical FFA to identify *single and plausible combinations of multiple functional* failures and assess their effects and criticality. This study can assist the development of an appropriate initial architecture for the system, which is then further refined as the system is decomposed into sub-systems and basic components. The result of this process in HiP-HOPS is a consistent *hierarchical model* that progressively records with increasing detail the implementation of the system (see Fig. 2, below *System Design*).

As the refinement of this hierarchical model proceeds, the failure behaviour of components in the model is analysed using a modification of classical FMEA called Interface Focused-FMEA (IF-FMEA). The application of this technique generates a *model of the local failure behaviour* of the component under examination, which is represented as a table. The table provides a list of component failures modes as they can be observed at the component outputs,

a) Identification and listing of all system functions
b) Precise definition of purpose and behaviour of each function
c) Examination of each function for potential failure modes in three classes:
   Loss of function
   Function provided when not required (commission)
   Incorrect operation of function
d) Determination of the effects of each failure. Wherever this is appropriate effects are determined in combination with other contributing factors [environmental factors, for example]
e) Determination of the severity of each functional failure
f) Compilation of the results in tabular form
   [*function, failure mode, contributing factors, effect, severity*]

Fig. 3. The standard FFA process.

and for each such *output failure*, it determines the causes as a logical combination of *internal malfunctions of the component* or *deviations of the component inputs.* An IF-FMEA table records how a hardware or software component reacts to failures generated by other components. In addition, the table determines the failure modes that the component itself generates or propagates to other components. As we shall show (in Section 5 and Section 7.3), this type of analysis can provide a useful resource in the design of the failure detection and mitigation mechanisms of the component under examination and other components in its periphery.

Once we have determined the local failure behaviour of all components, we can then proceed to the final stage of the analysis where we determine the structure of the fault propagation process in the system. At this stage, we determine how the functional failures that we have identified in the exploratory FFA arise from combinations of the low-level component failure modes that we have identified in the IF-FMEAs. In HiP-HOPS, this is achieved *mechanically* with the aid of a *systematic algorithm* for the *synthesis of fault trees*. Fault trees are constructed by exploiting the structure of the hierarchical model and information about the local failure behaviour of components that is contained in low-level IF-FMEAs.

The proposed fault tree synthesis algorithm is mechanical, and it was, therefore, possible to automate it. Indeed, in the context of this work we have implemented this algorithm as part of an experimental tool[3] that enables hierarchical modelling of the system and the safety analysis process defined by HiP-HOPS. In the following sections we examine in more detail the three stages of this process: *FFA*, *component failure analysis* and *fault tree synthesis*.

3. An extended FFA process

FFA is a relatively recent safety analysis technique, which is recommended by a number of standards as the *first step* in the assessment of new or modified complex systems. Fig. 3 records the main steps of a standard FFA process as it is defined in SAE ARP-4761 [1]. The process starts with the identification and listing of the system functions and continues with the precise definition of the purpose and behaviour of each function. Each function is then examined for specific failure modes in three general categories of failure: loss, inadvertent delivery and incorrect operation of function. For each identified failure, the analysis determines the effects on the system and the severity of failure.

This standard process encourages and systematises the anticipation of functional failures at the early stages of the design. The analysis, however, is restricted to *single* functional failures and it does not address issues of failure detection and recovery. This type of analysis is probably sufficient if we can make valid assumptions of independence between all system functions. In the general case, though, there will be dependent functions, for example functions that utilise common material, energy or information resources. In such cases, we need to address the possibility of multiple (dependent) functional failures. We believe that one way to explore functional dependencies is by constructing a more elaborate functional model than the list of functions currently used in standard FFA.

The model that we propose is a functional block diagram, which identifies functions and records how these functions rely on physical parameters or data. Fig. 4 illustrates an example of such a model for a system which delivers three functions (*A*,*B*,*C*). The model clearly identifies two functional dependencies. The first such dependency is between functions *A* and *B*. It can be noticed that the two functions operate on a common physical input (*p*). Any deviation of this input, therefore, is likely to cause a malfunction of both *A* and *B*. The second dependency that the model identifies is between functions *B* and *C*. Clearly function *C* operates on the output of function *A* and, therefore, relies on the correct operation of *A*. The construction of this functional model is the first step in an extended FFA process that we propose as part of HiP-HOPS and which we present here in Fig. 5.

[3] This tool is in fact an experimental extension of an existing commercially available tool for safety analysis called the Safety Argument Manager [8].

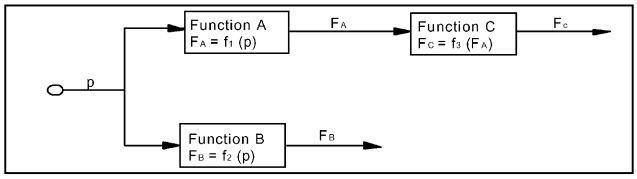


Fig. 4. Example functional model.

The first objective in this process is to identify and remove any avoidable dependencies between the functions that we have specified in the functional model. Each function is then systematically examined for potential failure modes in a number of abstract failure classes, which include the *loss* of function, the *unintended delivery* of function and *malfunctions* such as *early* or *late* deployment. For each identified failure the analyst determines the effects and severity of failure and lists the results in a tabular form. At this point, analysts are also expected to think of potential mechanisms for failure detection and recovery. We actually believe that it is possible to address such issues at the early stages of the design and we, therefore, extend the FFA table to include a *detection* and a *recovery* column.

Once we have identified all the single functional failures, we can then identify and list *plausible* combinations of multiple failures and, in a similar way, examine the effects and criticality of such failures. A difficulty that analysts may encounter here is the potentially large number of possible combinations between functional failures. The number of combinations that require examination can be constrained by exploiting symmetries in the functional model, by excluding combinations of failures that can only occur under mutually exclusive conditions and by applying other application specific plausibility criteria.

The results of this analysis are listed in a tabular form. As we demonstrate in our case study, they provide a comprehensive picture of the ways that the system can fail, and assist in focusing early the design effort to a number of important issues:

1. the prevention of hazardous single functional failures, by identifying and allocating critical functions to reliable fault tolerant architectures;
2. the prevention of multiple (dependent) functional failures, by removing avoidable dependencies between functions and developing partitions between the systems that deliver those functions;
3. the design of mechanisms for failure detection and recovery from single and multiple functional failures.

4. Hierarchical modelling

FFA assists the development of an appropriate initial architecture, which identifies the basic failure detection and fault tolerance strategies of the target system. During the design decomposition, this initial architecture is further refined as the system is typically decomposed into sub-systems, and then these sub-systems are decomposed into more basic architectural modules. In HiP-HOPS, the result of this process is a consistent *hierarchical model* of the system that progressively records with increasing detail the implementation of the system. To achieve consistency in this model, we place constraints on the modelling notations used, and introduce some additional notation for describing levels of design.

The notation allows complex systems to be modelled as

a) Construct a functional block diagram, which identifies system functions and their dependencies
b) Remove any avoidable dependencies between functions
c) Identify single functional failures examining each function for failure modes in a number of failure classes such as
   - the loss of function,
   - the inadvertent delivery of function, and
   - malfunctions such as early or late deployment, or the function output being excessively high, low, stuck etc.
d) Assess single functional failures
   - Determine any contributing factors (environmental factors, for example)
   - Determine the effects and severity of failure
   - Determine potential mechanisms for detection and recovery
   - Compile the results in a tabular form
     [*failure mode, contributing factors, effect, severity, detection, recovery, recommendations*]
e) Identify unique, plausible combinations (first pairs then triplets etc) of multiple functional failures
   - Identify which combinations are unique by examining any symmetries in the functional design
   - Exclude those combinations that can only happen under different and mutually exclusive conditions
   - Further constrain the number of combinations to be examined by applying other plausibility criteria
f) Assess multiple functional failures
   - As in step (d)

Fig. 5. The proposed extended FFA process.

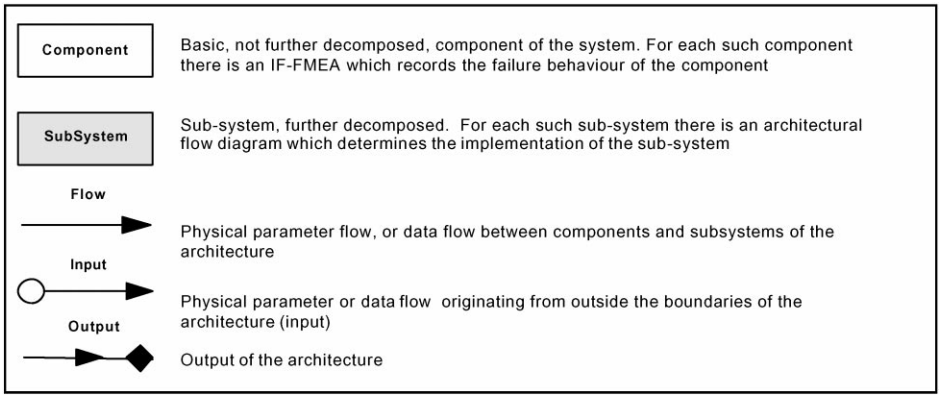


Fig. 6. Modelling notation.

hierarchies of architectural diagrams. At each level of the system hierarchy, flow diagrams are used to describe the operation of the system and its subsystems. At plant level, such flow diagrams can be derived, for example, from engineering schematics or piping and instrumentation diagrams. At lower levels they can be derived from any form of structured design notation used for the architectural design of software or hardware components, for example Data-flow diagrams [9] or MASCOT diagrams [10]. The five primitive elements of the proposed notation are illustrated in Fig. 6.

According to the notation, modules of an architecture can be represented either as *components* or *subsystems*. If the failure behaviour of a module is known, and it can be recorded in an IF-FMEA table, the module is represented as a basic *component.* In the opposite case, the module is rendered as a *subsystem*, and is further decomposed into an architecture of more basic *components* the failure behaviour of which can be determined using IF-FMEA. *Components* and *subsystems* communicate exchanging flows, which represent the material, energy or data transactions between the elements of the architecture. To enable this type of modelling, we have implemented in the Safety Argument Manager a *hierarchical model editor* that supports the proposed notation (see Fig. 7).

5. Assessment of failure at component level

As the refinement of the hierarchical model proceeds, we identify basic hardware and software components. The failure behaviour of these components is analysed using an extension of FMEA. Traditional FMEA examines the

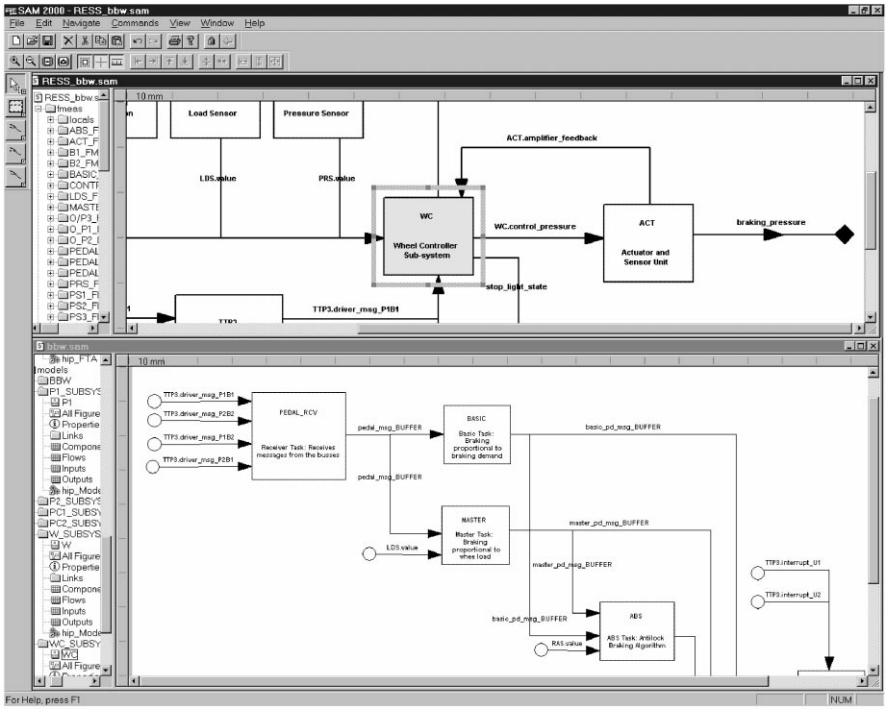


Fig. 7. The hierarchical modelling editor.

origins of failure within the component itself. In other words, it examines the failure behaviour of the component considering only internal malfunctions (possibly caused by the environment). The function of a component in the failure domain, however, is significantly more complicated. A component does not only generate failure events. It can also detect (or not) and respond (or not) to failure events generated by other components which interface to the component via its inputs.

A component, for example, may detect disturbances of its input parameters, e.g. the absence of a power signal, or a value that is out of range. In turn, the component can *mitigate* the propagation of such failure events. In the absence of a power signal, for example, the component may automatically switch to a redundant power supply.[4] Similarly, it may replace a detected invalid input value with a correct default value. A component can also fail to detect such input failures and it can *propagate* those failures to other components. Finally, it may *transform* a certain input failure event, to a different type of output failure. An example of such a component is the TTP/C communication controller that is used for the exchange of messages between the nodes of the brake-by-wire system. The controller detects the early or late delivery of messages from the host and, in response, it cancels the transmission of those messages to ensure the integrity of other data on the bus [11]. This clearly represents a case of a *timing failure* being transformed into an *omission* failure.

To capture those additional aspects of the behaviour of the component in the failure domain, we propose an extension of FMEA, called IF-FMEA (*Interface Focused-FMEA*). IF-FMEA is a tabular technique, which can be used in a similar way to traditional FMEA in order to examine component failure modes caused by internal component malfunctions. Beyond that, however, the method provides a systematic way to examine the *detection, mitigation and propagation* of failure across the component *input* and *output interfaces*. The method requires a model of the component, which identifies the component inputs and outputs. In the course of the analysis, each output of the component is systematically examined for potential deviations from the intended normal behaviour. The specific failure modes of each output are determined as the behaviour of the output is scrutinised for potential deviations that may fall in one of the following three abstract categories of failure:

1. *service provision* failures such as the omission or commission of the output;
2. *timing failures* such as the early or late delivery of the output;
3. failures *in the value domain* such as the output value being out of a valid range, stuck, biased, exhibiting a linear or non-linear drift or erratic behaviour. [5]

The result of this analysis is a model of the local failure behaviour of the component under examination. This model is represented as a table and it provides a list of component failures modes as they can be observed at the component outputs. For each such *output failure* the analysis determines the causes as a logical combination of un-handled *internal malfunctions of the component* or un-handled *deviations of the component inputs*.[6] For each internal malfunction, the analysis records an estimated or experimentally derived failure rate λ in failures per hour (f/h) or other suitable units.

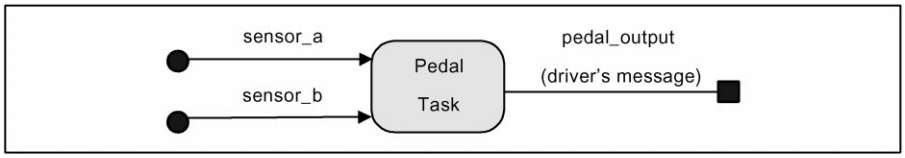
In HiP-HOPS, this technique replaces the traditional failure mode and effects analysis of hardware components. Beyond hardware safety analysis though, IF-FMEA is also used for the analysis of software architectures. In the case of a software module, a task for example, the IF-FMEA table records how the task responds to omission, commission or corruption of input data caused by *provision*, *timing* and *value failures* propagated by other software modules. In addition, the table shows how malfunctions of the task can be caused by failures of subsidiary elements such as the processor, the operating system or the memory elements used by the task. An example of software IF-FMEA is illustrated in Fig. 8.

The figure contains a fragment from the IF-FMEA of the pedal task, which is located on the pedal node of the brake-by-wire system. The task reads the braking demand provided by two pedal sensors, detects invalid (out of

[4] The component, for example, is connected to the second power supply through a normally open contact which is relayed to the main power signal. As long as power is received from the primary supply, the contact remains open and breaks the secondary circuit. In the case of power failure, the contact closes and the component automatically connects to the secondary power source.

[5] The basis of this categorisation can be found in earlier work on classifications of failures in particular domains (see [12–14]). In [14] for example, McDermid and Pumfrey propose a HAZOP inspired technique for software hazard analysis in which the above classification (*provision, timing* and *value failures*) guides the identification and assessment of hazards in software architectures. Their original classification distinguishes between two broad classes of value failures: subtle and coarse failures. Although we have essentially adopted this classification for the purposes of IF-FMEA, whenever this is appropriate we attempt a separation of value failures in more detailed classes (such as value *out of range*, *stuck*, *biased* etc). As we demonstrate in the brake-by-wire case study (Section 3.3.3), a more detailed classification of value failures can increase the information content and the value of the analysis and can help to identify appropriate techniques for detecting failures that belong to particular classes.

[6] We must note that our work on IF-FMEA draws from the work of Fenelon [15,16] on the Failure Propagation and Transformation Notation, a graphical notation for the representation of the transformation and propagation of failure in a system. Although an FPTN module and an IF-FMEA table differ in form, they are conceptually similar in the sense that they both define a set of equations which characterise the logical relationships between input and output failure events. In contrast to FPTN modules though, IF-FMEAs are semantically and syntactically linked to the design representation of the system. Input and output failure events, for example, always represent deviations of parameters that are explicitly represented as flows in the hierarchical model of the system, and they strictly refer to those parameters with the name that they possess in the model. As we shall show in the following section this link enables the mechanical generation of fault trees from the design representation and the IF-FMEAs of its components.



Output Failure Mode	Description	Input Deviation Logic	Component Malfunction Logic	+ (f/h)
O-pedal_output	Omission of *Pedal* output (driver's message). It can be caused by task malfunction or out of range failures of both pedal sensors.	(V>max-sensor_a | V<min-sensor_a) & (V>max-sensor_b | V<min-sensor_b)	processor_ failure | operating_ system_ failure	1.00E-07| 9.00E-07
Vs_0-pedal_output	*Pedal* output (driver's message) stuck at 0. It can be caused by memory stuck at 0 failures, or by stuck at minimum failures of both pedal sensors.	Vs_min-sensor_a & Vs_min-sensor_b	Memory_ stuck_at_0	2.00E-06

Fig. 8. Model and fragment of the IF-FMEA of the pedal task.

range) measurements and provides as output the average of the valid sensor readings. The output of the pedal task is the driver message that is send over the bus to the wheel nodes (see Fig. 1). The IF-FMEA table examines two potentially hazardous failure modes of this output.

The first such failure is an omission of the driver message. The analysis shows that this event can arise from a task malfunction with an *assumed* overall failure rate of $10^{-6}(10^{-7} + 9 \times 10^{-7} = 10^{-6}$ f/h). Such a malfunction can be caused by a failure of the processor ($= 10^{-7}$ f/h) or a failure of the operating system ($= 9 \times 10^{-7}$ f/h). The output event can also be caused by deviations of the inputs that the task receives from the pedal sensors. Indeed, if both sensor readings are out of the valid range of measurement, the task is unable to produce the driver message. The second hazardous output failure examined in the table is a condition where *the value of the driver message is stuck at zero* and does not represent the actual pedal stretch. This condition can be caused by stuck at zero failures of the memory elements used by the task with a failure rate of 2×10^{-6} f/h. In addition, the analysis shows that the condition will also occur if both sensor inputs are stuck at the minimum of the normal measurement range. Note that the above failure rates are only given as examples and do not represent reliability data of a real system.

Clearly, an IF-FMEA of a component shows how the component reacts to failures generated by other components. The table for the pedal task, for example, shows how the task responds to failures generated by the two sensors. In addition, the table determines the failure modes that the component itself generates or propagates. Such a table, we believe, can be employed usefully by the designer of the system in order to improve the failure detection and mitigation mechanisms of the component under examination and other components in its periphery. To enable this type of analysis and integrate it with the hierarchical model of the system, we have developed in the Safety Argument Manager tool a tabular (spreadsheet like) editor (Fig. 9).

6. An algorithm for the mechanical synthesis of fault trees

IF-FMEAs contain expressions that describe the causes of output failures as logical combinations of internal component malfunctions *or* deviations of the component inputs. These expressions are described in two columns of the IF-FMEA table: the *Input Deviation Logic* column and the *Component Malfunction Logic* column. The semantic relationship between the expressions in those two columns is a disjunction (i.e. a logical OR). Thus, given an arbitrary row of the IF-FMEA table, the logical relationship between the *Output Failure Mode* described in this row and the corresponding *Input Deviation* and *Component Malfunction Logic* is:

Output Failure Mode

$=$ *Input Deviation Logic* | *Component Malfunction Logic*

where | *represents the disjunction operator* (*OR*)

The grammar of a combined expression that determines the causes of output failure in the above form is given in Fig. 10 (described in Extended Backus Naur Form). This grammar

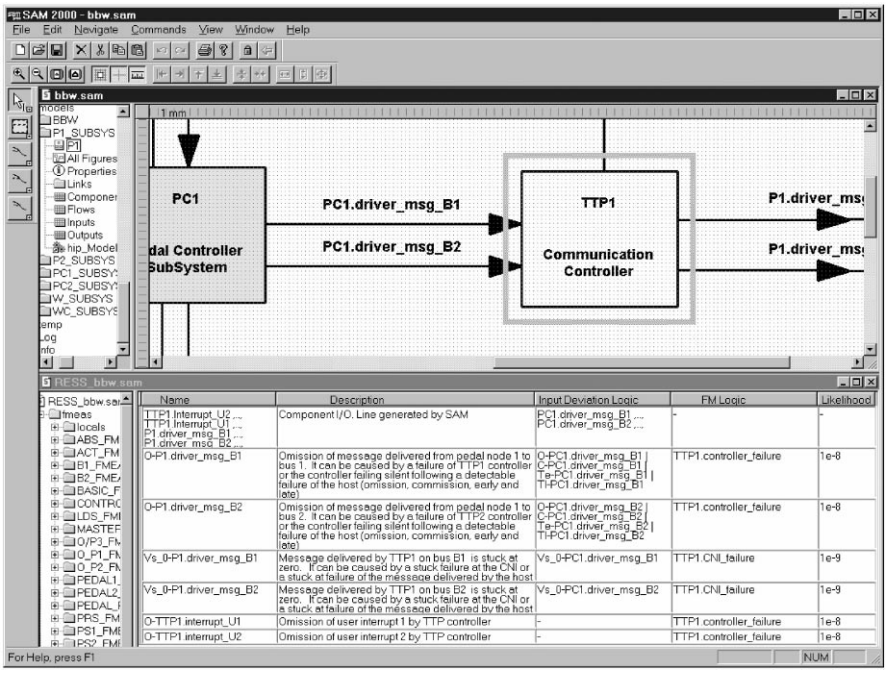


Fig. 9. Component and its safety analysis (IF-FMEA).

recognises parenthesised logical expressions that contain conjunction (&) and disjunction (|) operators and other terminal symbols representing *component malfunctions* or *input deviations*. Such terminals are recognised as *component malfunctions*, unless they contain a hyphen (-) in which case they are recognised as *input deviations*. The part preceding the hyphen is interpreted as the *failure class* (‘O’ for omission, ‘C’ for commission, ‘Vs_0’ for value stuck at zero etc.) while the remaining part of the string is interpreted as the name of the component input.

The grammar of Fig. 10 is right recursive and gives precedence to conjunction operators. Thus, for example, the expression a|b&c is interpreted as a|(b&c) and not as (a|b)&c. In terms of syntactical analysis, the grammar can be interpreted by a relatively simple and efficient parsing scheme. This scheme is, in fact, a top–down predictive parser [17] that can perform syntactic analysis and interpretation of expressions that conform to the foregoing grammar without backtracking through production rules in the course of the translation process.

Fig. 11 shows the parse tree generated when, for example, the parser processes the expression:

$$short\_circuit \mid mechanical\_failure \mid (O - input1 \mid O - input2) \,\&\, O - power.$$

It can be noticed easily that the logical structure of this parse tree is identical to that of a mini fault tree that represents in graphical form the failure logic described in the expression. Fig. 12 illustrates the equivalent mini fault tree for the expression. We can clearly notice the identical position of logical connectives within the two

```
Causes_of_Output_Failure        ::= Or_Term { | Or_Term }
Or_Term                         ::= And_Term { & And_Term }
And_Term                        ::= Cause | ( Causes_of_Output_Failure )
Cause                           ::= Deviation | Comp_Malfunction
Deviation                       ::= Failure_Class - Input_Name

where

Failure_Class      The failure class
Input_Name         The name of a component input
Comp_Malfunction   The name of a component malfunction
```

Fig. 10. Grammar for IF-FMEA expressions.

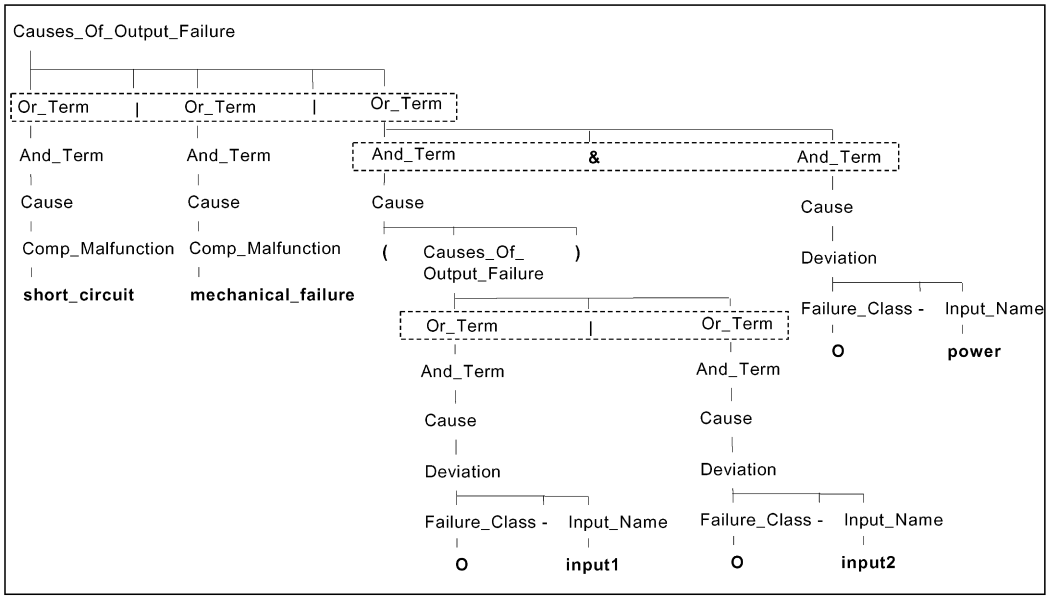


Fig. 11. The parse tree for the given example expression.

tree structures. The relationship between the parse tree and the fault tree is not coincidental nor is it limited to the above example. It reflects a generic property of the parser, which enables the simple and straightforward synthesis of a mini fault tree for any expression that conforms to the proposed grammar, in the course of the translation process.

In HiP-HOPS this mechanism is used as part of an algorithm for the automatic synthesis of fault trees. Fault trees for hazardous functional failures, as they are observed at the outputs of the system or its subsystems, are constructed by traversing the hierarchical model of the system and by following the propagation of failure backwards from the final elements of the design (actuators) towards the system inputs (sensors). The fault tree is generated incrementally as we parse the expressions contained in the IF-FMEAs of the components that we encounter during the traversal, and as we progressively substitute the input deviations received by each component with the corresponding output failures propagated by other components. The synthesis algorithm proceeds in two dimensions:

1. vertically, translating system (or sub-system) failures to component failures;
2. horizontally, translating output failures to combinations of component malfunctions and deviations of component inputs.

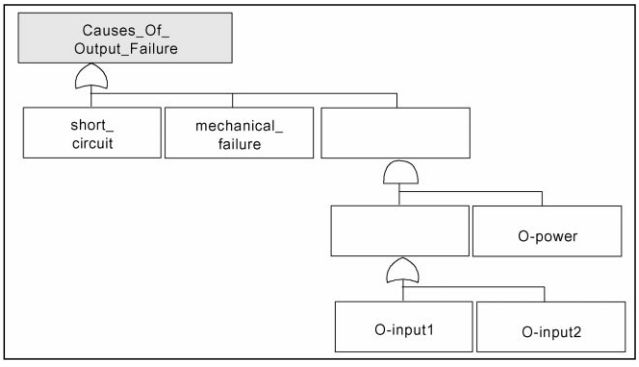


Fig. 12. The equivalent mini fault tree.

The search across the vertical axis of the hierarchy has always a higher priority. The rule is that when a sub-system is encountered during the traversal of the hierarchical model, the causes of its output failure are always traced first at the sub-ordinate hierarchical level of the design, which describes the architecture of the sub-system.

To integrate the fault tree synthesis into the modelling and safety analysis process, we have implemented the synthesis algorithm in our tool and integrated it with the modelling and IF-FMEA editors. The tool can now synthesise fault trees for functional failures at the outputs of the system or its sub-systems by traversing the system model and parsing the IF-FMEAs of its components. In practice, when we wish to generate a fault tree, we select an output parameter or an internal flow in the model and specify a deviation of this parameter from its intended behaviour (see Fig. 13). In response, the tool generates the fault tree for the given deviation. During the synthesis, the Safety Argument Manager derives the probabilities of component malfunctions from IF-FMEAs, and uses these data to perform minimal cut-set analysis and probabilistic calculations on the fault tree.

The fault tree is drawn on a scrollable canvas upon which we can zoom to study different parts of a large and complex fault tree. During the synthesis, the tool hyperlinks nodes of the fault tree to component renderings. Thus, it enables

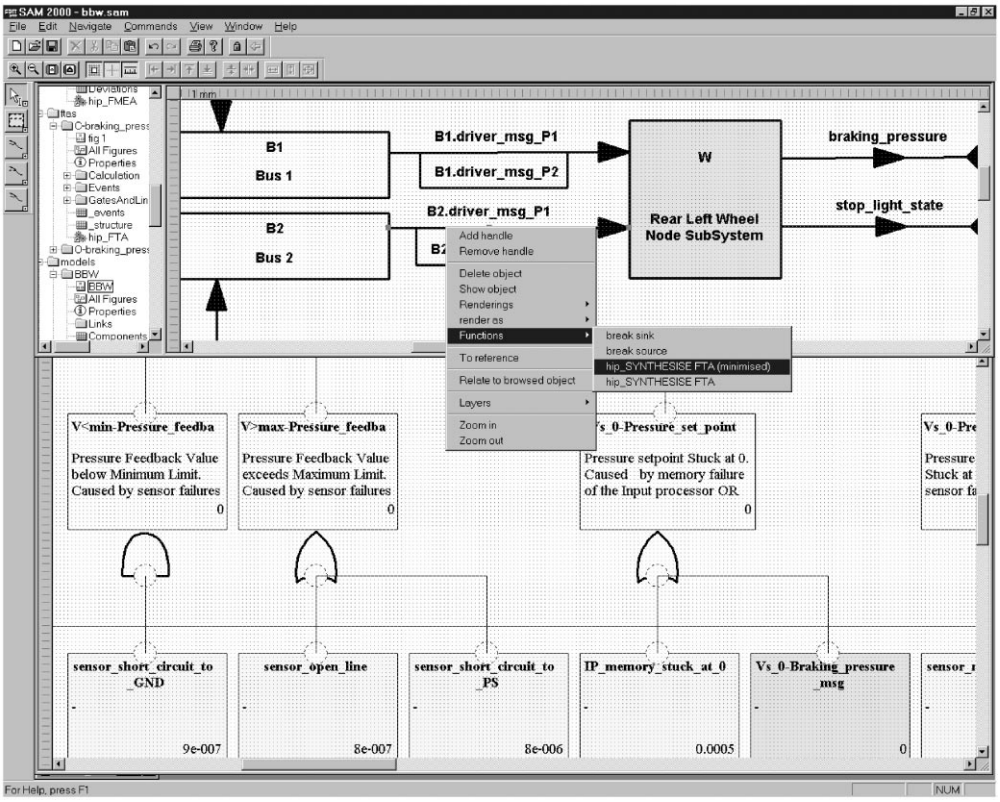


Fig. 13. Tool support for automatic Fault tree synthesis.

direct navigation between nodes of the fault tree and the hierarchical model of the system, so that we can always trace easily the origins and propagation of failure back to the system design. This aids review of the fault trees, and helps gain confidence that the mechanically constructed trees represent the failure behaviour of the design.

The synthesised fault trees record the propagation of failure in a very strict and methodical way. They start from failures of the final elements of the design (actuators) and following the dependencies between components in the model, they progressively record other component failures and deviations of the system inputs. The rules that we apply to capture the fault propagation in the tree structure are explicit, consistent and always the same. The resultant fault trees, therefore, have a logical structure which is determined by the application of these rules (synthesis algorithm), the interconnections between the components of the model and the IF-FMEAs of those components. This logical structure is straightforward and can be easily understood, unlike the structure of many manually constructed fault trees, which is often defined by implicit assumptions made by analysts.

For those reasons, we believe that the synthesised fault trees are easier to interpret in the context of the system model, and that such fault trees can provide a useful resource in the system design. The probabilistic calculation on these fault trees can indicate if the target failure rates for the critical functions of the system have been met. Cut-sets of the fault trees that contribute more to the overall failure probability can directly point out weak areas of the design and initiate useful design iterations. Currently, re-design creates enormous difficulties in the maintenance of large manually constructed fault trees. In contrast, design iterations would not pose problems to the synthesis of the fault trees as new fault trees could be re-constructed automatically after each design iteration after certain changes in the underlying safety analyses.

The idea underlying the proposed fault tree synthesis is a simple one and it can be summarised in the following proposition:

> If we know the “structure” of a system (model) and the “local failure behaviour of its components” (IF-FMEAs) then we can mechanically derive the “failure behaviour of the system” (fault trees).

This proposition implies that the synthesis algorithm could also be used for the mechanical synthesis of *system IF-FMEAs*. Indeed, to synthesise an IF-FMEA for a system one would just need to construct the fault trees for all the possible deviations at the system outputs, and then derive the minimal disjunctive form of those fault trees. Such IF-FMEAs could help in rationalising and simplifying complex safety assessments, as they would provide failure models which could be re-used within the same application or even across different applications, possibly after some

necessary minor adjustments to reflect the effects of a different environment.

Finally, we must point out that the quality of the mechanically generated fault trees is strongly contingent to the quality of the system model and the low-level analyses of its components. But while the method does not necessarily generate perfect fault trees, on the other hand it ensures that the assumptions underlying the structure and content of those trees (i.e. model and IF-FMEAs) have been recorded and they are in an explicit, complete and accessible form for reviews. This, in our view, is probably the most significant contribution of the synthesis algorithm in improving the quality of the results from the safety assessment.

7. Case study

Our case study on the brake-by-wire system is a safety case that we have produced in parallel to the development of this system using HiP-HOPS and the Safety Argument Manager tool. The study is based on a detailed hierarchical model of the implementation of the system. Our analysis starts at the functional level of the design, and proceeds all the way down the design hierarchy to examine the failure behaviour of low-level hardware components such as sensors and actuators and software components such as the tasks running on the nodes of the distributed architecture.

In this section we discuss the results from this study and highlight our most significant findings. Drawing from these results, we will attempt to illustrate how HiP-HOPS has helped us to overcome some of the problems of classical techniques, improve the system design, and how at the end of the assessment process we have achieved a consistent and meaningful safety case.

7.1. FFA

The FFA of the brake-by-wire system was carried out early in the design process. Although it was based on a conceptual design of the system, it helped us to determine ways in which the system can fail beyond the obvious loss of braking, and to consider detection and recovery mechanisms for these failures. Fig. 14 illustrates the abstract functional model of the brake-by-wire system which provided the basis for the analysis. The model shows the four braking functions delivered by the system (one on each wheel), their input parameters and their outputs.

Our first observation on the model is that the *four braking functions are dependent* since they rely on the same physical input, the *braking demand* provided by the driver. This dependency is essential and it cannot be removed. The implication for the design is that the braking demand must be carried in a reliable manner from the driver's pedal to the

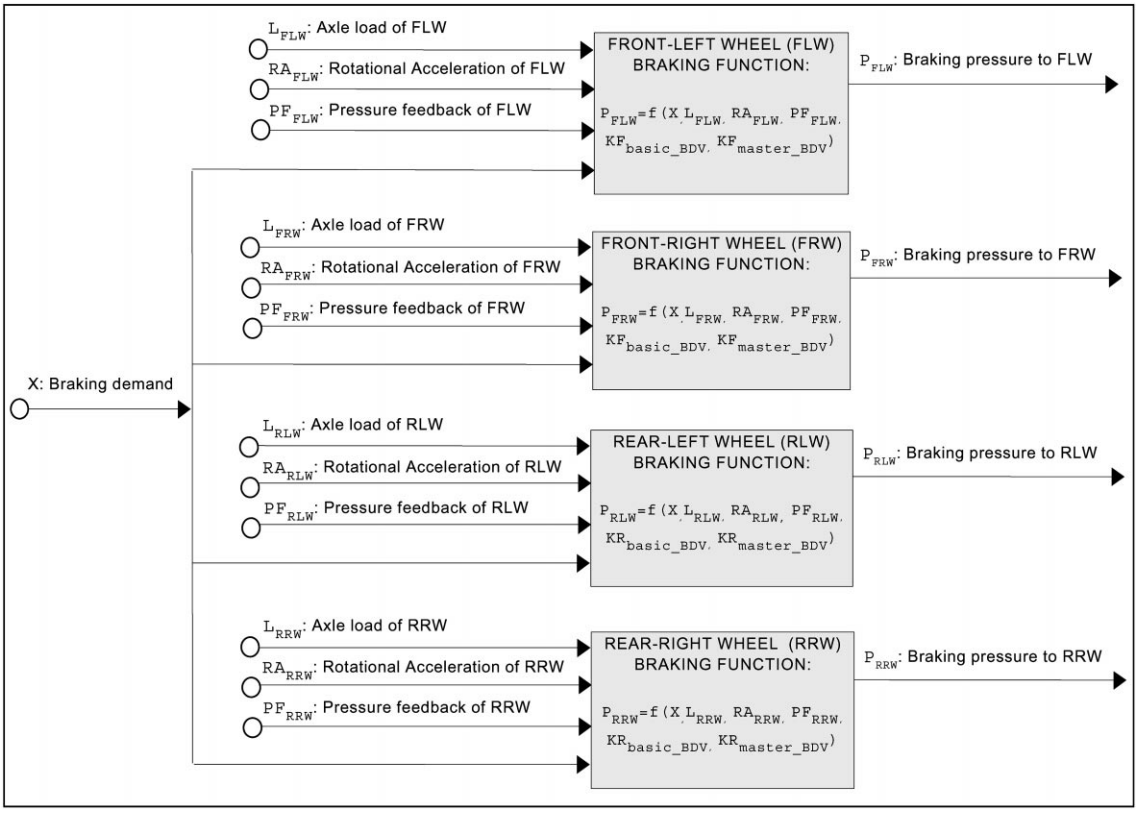


Fig. 14. Abstract functional model of the brake-by-wire system.

wheels. We have, therefore, decided early to replicate the pedal nodes, which capture the braking demand and use a dual bus for carrying this information to the wheel nodes. The model also shows that the four braking functions are almost identical. They perform identical calculations on identical *types* of input data (braking demand, axle load, wheel rotational acceleration, and pressure feedback) and generate identical types of output (braking pressure to the wheel).

Their only difference is defined by the parameters KF and KR which determine the distribution of braking between the front and rear axle of the car (the ratio *front:rear* is approximately 60:40). This symmetry of the functional design across the longitudinal axis of the car made it possible to examine only two out of the four braking functions in the course of the FFA.

The first part of the FFA identifies *single* functional failures of the *front-left* and *rear-left wheel* braking functions. Each of the two functions has been systematically examined for potential failures in a number of failure classes, which include the loss of function, the unintended delivery of function and malfunctions such as early or late deployment. For each failure that we have identified, the analysis records the effects in terms of the impact on three parameters of the car: stability, steerability and braking capacity. The severity of each failure is described using the standard four severity classes of IEC-61508 [18] (catastrophic, critical, marginal, insignificant). Our analysis shows that certain functional failures can be mapped to other types of failures. *Late* and *less braking*, for example, can be seen as temporary *loss of braking* with obviously less severe effects than permanent loss of braking. Table 1 summarises the four most severe failures of each braking function.

It can be noticed that in this type of distributed system there is no *catastrophic* single functional failure and that all single functional failures are tolerable (although *critical*). There are also possibilities for detection and recovery from a number of failures. Some of the effects of a permanently locked wheel on the stability of a car, for example, can be compensated by a recovery function which in response to the initial failure locks the diagonal wheel (see Table 1; FL7).

Table 1
Main functional failures of a wheel braking function

Failure	ID	Effects on System	Severity	Detection	Recovery	Recommendation
Loss of Braking (omission) When there is braking intention	FL3	The car tends to drift to the side −30% stability −18/−32% braking (rear/front) −15% steerability In the worst case the drift is opposite to the drivers intention	Critical	Locally, using feedback from a pressure sensor Remotely, by a local status reporter and a remote monitor task	Not Possible	In addition, the failure can be detected by a global rotational acceleration sensor. An Electronic Stability Program device may handle the problem (this is out of the scope of this brake-by-wire system)
Unintended Braking (Commission) When there is no braking intention	FL4	The car tends to drift to the side	Critical	It is possible in certain cases by comparing the state of the pedal with a pressure sensor feedback from the wheel	Release actuator	The detection algorithm should be sufficient to detect pedal sensor failures and internal corruption of the pedal messages. There should be provisions to keep commission failures temporally limited
Permanently Locked Wheel When there is braking intention	FL7	The car tends to drift to the side −30% of stability −1/−2% of braking (rear/front) −65% steerability In the worst case the drift is opposite to the drivers intention	Critical	It is possible, by using feedback from a rotational acceleration sensor	Release pressure until wheel unlocked If for any reason the wheel remains locked, then lock the diagonal wheel	Incorporate an ABS algorithm, to prevent permanent locking of wheel. In addition implement a detection mechanism for ABS failure and a recovery mechanism at system level (locking of diagonal wheel)
Permanently Locked Wheel When there is no braking intention	FL8	Equivalent to FL4 but more severe since maximum braking is applied	Critical	It is possible, see FL4 and additionally by using feedback from a rotational acceleration sensor	See FL4	See FL4

In the second part of the FFA, we have considered combinations of multiple (two, three and four) functional failures. The first step here was to identify the plausible combinations of such failures. The system has four braking functions and each function has four major failure modes (see Table 1). Although the number of the possible combinations of those failures is large, a systematic examination of those combinations has shown that the number of *unique* combinations is relatively small. One reason is the symmetry in the functional design of the brake-by-wire system. In addition, certain combinations are impossible because they can only occur in different and mutually exclusive conditions (braking intention/no braking intention).

The analysis has shown that, with the exception of loss of rear axle braking and loss of "diagonal braking", the loss of two or more braking functions is intolerable (severity = *catastrophic*), mainly because of unacceptable loss of braking capacity. Recovery is impossible and, therefore, such multiple failures shall be prevented by design, for example by taking measures against common cause failure. The analysis also indicates that the commission of two or more functions (unintended braking) can be tolerable if it is temporally limited, in other words if it is a result of short transient failures or if there is rapid detection and release of the braking actuator.

Finally, the analysis shows that wheel locking affects the stability and steerability of the car. The severity of the effects varies from *marginal* (two diagonal wheels), to *catastrophic* (rear axle, three wheels). An interesting observation is that while the severity of a single locked wheel is *critical*, the severity of two locked diagonal wheels is *marginal*. This indicates that it is possible to use the intentional locking of the diagonal wheel as a simple recovery from a single wheel locking failure (see Table 1:FL8). The analysis also confirms that the locking of four wheels is less severe (*critical*) than the locking of three or, in some cases, two wheels. Thus, intentional locking of all wheels can be used as a compensatory mechanism against certain more serious locking failures.

7.2. Hierarchical model

This high-level FFA equipped us with a comprehensive view of the ways in which the system can fail, and helped the design of the initial architecture of the system. To ensure consistency between the analyses, all the remaining aspects of safety assessment were performed on a consistent *hierarchical model* of this architecture. The hierarchy was developed during the design decomposition process. The process involved the decomposition of the system into sub-systems, and then further decomposition of each subsystem into more basic modules. The hierarchical model that we have developed in the Safety Argument Manager tool is precise and is based on the actual hardware and software implementation of *one* braking function over two pedal nodes and one wheel node.

Fig. 15 illustrates the top-level architecture of this model. It can be noticed that the pedal nodes (P1, P2) and the wheel node (W) are marked as sub-systems. Each such sub-system in the model has a subordinate hierarchical level, which describes the architecture of this sub-system. This decomposition process is repeated until we reach the low levels of the hardware and software implementation. Fig. 16, for example illustrates two successive layers in the architectural decomposition of the wheel node sub-system (W).

In the first layer, we see the architecture of the wheel node, composed of a communication controller, an actuator unit, three sensors (rotational acceleration, load and

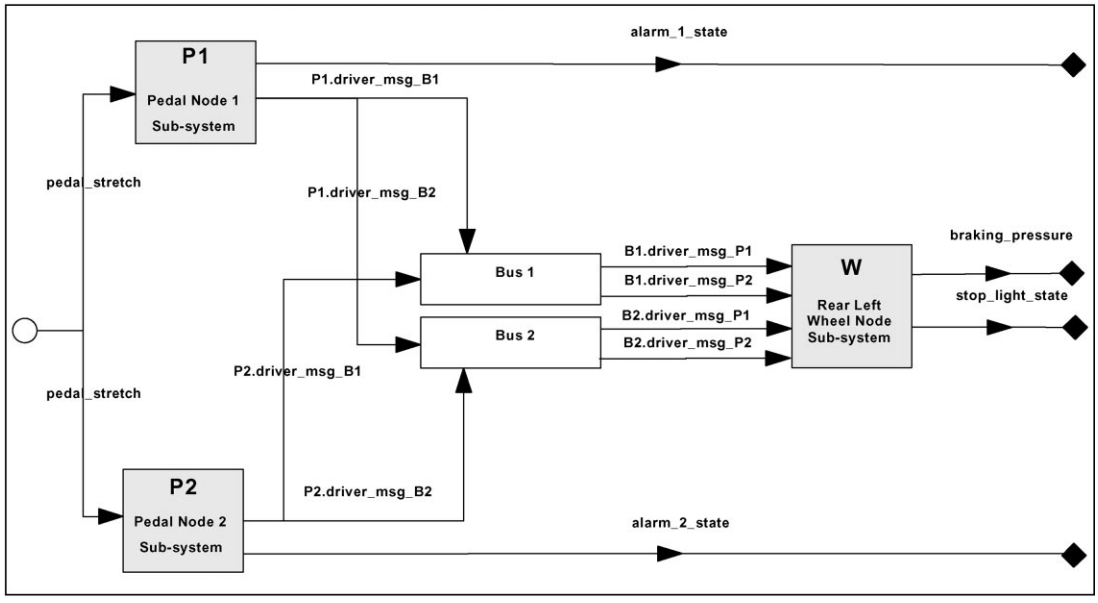


Fig. 15. Top level of the hierarchical model of the brake-by-wire system.

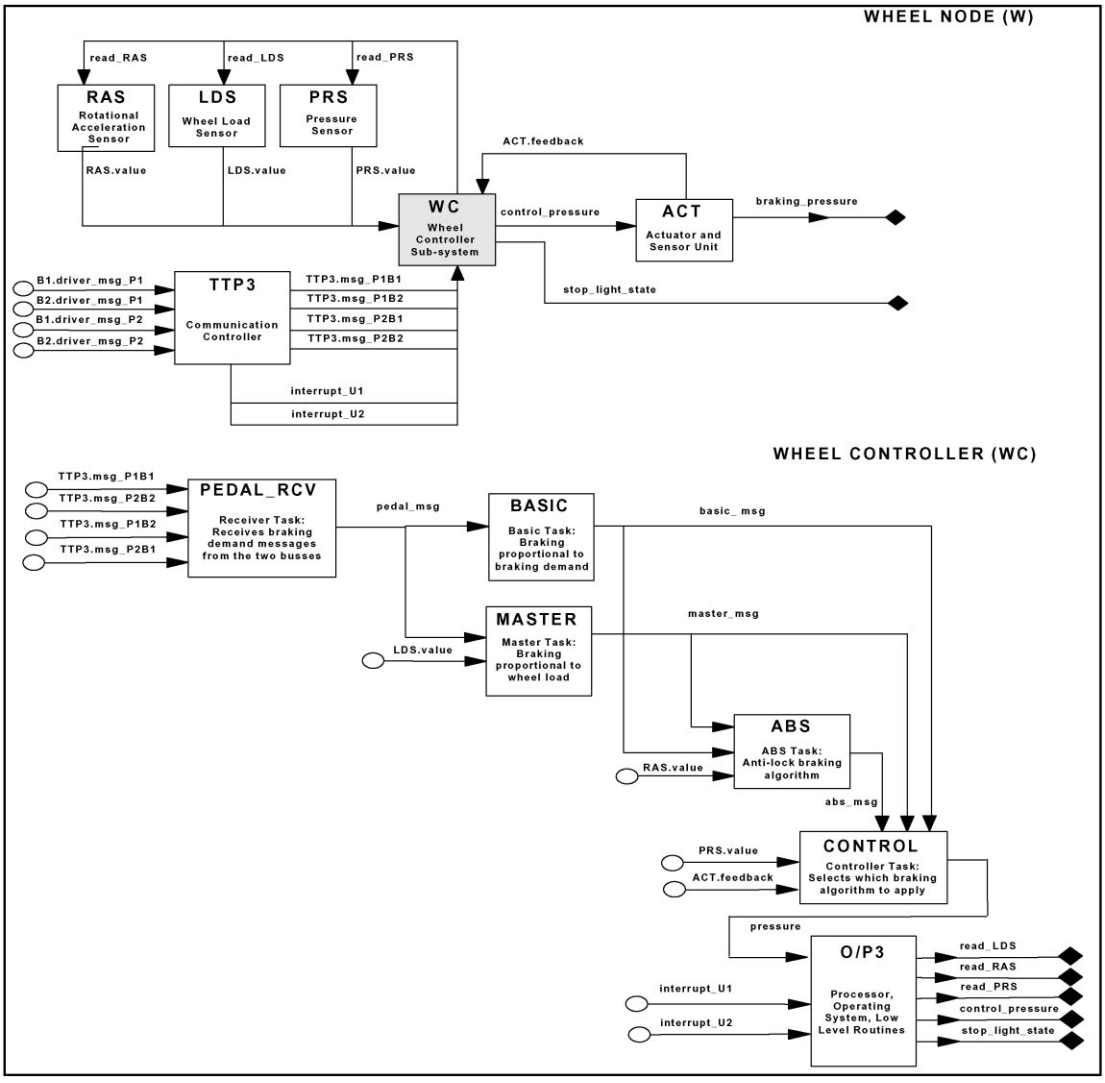


Fig. 16. Two successive levels in the hierarchy of the brake-by-wire model.

pressure feedback sensor) and a wheel controller. At the second level of the decomposition, Fig. 16 illustrates the architecture of the wheel controller. The model at this level identifies the software tasks that compose the software architecture of the wheel controller as well as the data flows between them. We must point out that the intention of those figures is rather to provide an indication of the complexity of the model than to show details of the design.

7.3. Component safety analyses

As the hierarchical model of the system evolved, IF-FMEAs contributed further to its improvement. Sensor IF-FMEAs, for example, provided lists of sensor failure modes as these can be observed at the sensor outputs (e.g. output stuck, biased, out of range, exhibiting non-linear drift etc.). These IF-FMEAs have directly supported the design of hardware redundancy schemes and averaging or voting algorithms for the detection and masking of sensor failures.

Fig. 17 illustrates an example of a sensor/converter unit. The unit receives one input, the signal that triggers the *analogue to digital* conversion, and delivers one output, the measurement value. The IF-FMEA of this unit is presented in Table 2. The table lists all the potential failure modes that can be observed at the output of the component. In other words, it provides a detailed account of the failures that the component generates and that it potentially propagates to other components. Let us now see how this table can assist the design of a component that interfaces to the sensor and operates on the measurement that the sensor provides at its output.

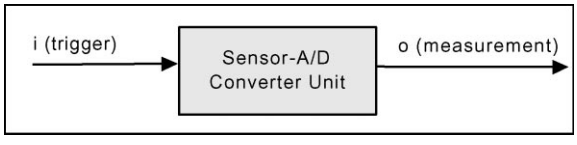


Fig. 17. Sensor and analogue to digital converter unit.

Table 2
Sensor IF-FMEA

Output failure mode	Description	Input deviation logic	Component malfunction logic	λ (f/h)
$V_{v<MIN} - o$	Output value below valid range. Caused by a short circuit to ground (e.g. because insulation has failed) or by noise when the value is close to the minimum of the measurement range.	–	Short_circuit_to_ground|Noise	1E-4|2E-4
$V_{v>MAX} - o$	Output value above valid range. Caused by a short circuit to power supply or by open line or by noise when the value is close to the maximum of the measurement range.	–	Short_circuit_to_power_supply|Open_line|Noise	1E-4|2E-4|1E-5
$V_{v\_DRIFT} - o$	Output drifts from actual measurement. Caused by noise, high temperature or more subtle mechanical problems	–	Noise|Temperature|Mechanically biased	2E-4|1E-4|1E-3
$V_{v\_STUCK} - o$	Output stuck at a certain value. Caused by mechanical parts being stuck or by memory buffer failure	–	Mechanically_stuck|Memory_buffer_stuck	1E-3|1E-5
$O - o$	Failure to update the sensor value buffer. Caused by a converter failure or by an omission of the trigger signal (e.g. trigger line is broken). Similar effect to $V_{v\_STUCK} - o$	$O - i$	Converter_failure	1E-5
$C - o$	Inadvertent update of sensor value buffer. Caused by a converter failure, electromagnetic interference (e.g. lightning) or by a commission of the trigger signal	$C - I$	Converter_failure |Electromagnetic_interference	1E-5|2E-5
$E - o$	Early delivery of measurement Caused by a timing failure of the converter. Similar effect to a commission failure ($C - o$)	–	Converter_failure	1E-5
$L - o$	Late delivery of measurement. Caused by a timing failure of the converter	–	Converter_failure	1E-5

Let us assume for example, that the sensor of Fig. 17 is a pedal sensor, and that we want to design a *pedal* task which reads the braking demand and then broadcasts this value on the dual bus. The failure rates in the IF-FMEA of the sensor suggest that the sensor is a fairly unreliable component. To capture the braking demand in a fairly reliable manner we, therefore, need to employ at least two sensors. The table also indicates that the pedal task should perform range checks on the value of the monitored variable in order to detect invalid (*out of range*) measurements ($V_{v<MIN} - o$ and $V_{v>MAX} - o$) that can be caused by a number of sensor failures.

It can be noticed that noise may also cause invalid (*out of range*) indications, when the value of the parameter lies close to the minimum or maximum of the measurement range. In the worst case noise will shift a value, which reflects a demand for maximum braking into the region of invalid measurements. This in turn may cause a failure to brake. To prevent hazardous false alarms arising from noise in border conditions, the pedal task should consider the possible presence of noise in its interpretation of the sensor output. A simple way to achieve this is by relaxing the range of valid measurements to tolerate a reasonable level of noise. Beyond noise, high temperature and mechanical problems can also cause small drifts, which distort the measurement of the physical parameter ($V_{v\_DRIFT} - o$). One way to compensate against such drifts is by averaging the valid sensor values.

It is important to point out that this strategy also offers some protection against *stuck at* ($V_{v\_STRUCK} - o$) and omission ($O - o$) failures of the sensors. Indeed, the average of the two measurements will be *stuck at* a certain value only if both measurements are stuck at the same or different values. The pedal task, therefore, will fail to deliver a non-zero braking demand only if both sensors are stuck at zero. Indeed, if only one measurement is *stuck at zero*, the

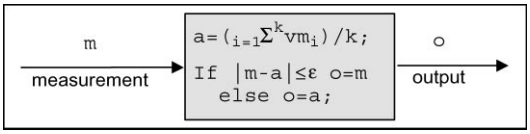


Fig. 18. Simplified model of the peak detection and removal task.

pedal will average zero with the correct measurement and send to the wheel nodes a non-zero value which represents half of the actual braking demand.

IF-FMEA also contributed in the analysis of software architectures and the identification of subtle errors in certain software algorithms. The technique has helped us, for example, to identify a condition that could make an early version of the generic *peak detection* algorithm of the brake-by-wire system fail with possibly severe consequences for the system. The peak detection algorithm is applied to sensor readings in order to detect and remove transients that violate the normal dynamic behaviour of the physical parameter. The average of the k last valid readings ($vm_1 \ldots vm_k$) is calculated in every cycle of measurement. If the current reading (m) deviates from this average more than a maximum allowable limit (ε), i.e. if

$$|m - \left(\sum_{i=1}^{k} vm_i\right)/k| > \varepsilon,$$

then it is considered invalid and is discarded. The model of a software task that uses such a mechanism is illustrated in Fig. 18. The task performs peak detection on the input value (m). When the input value does not violate the peak detection criterion, it is copied to the task output (o). In the opposite case, the output carries the average of the last k valid values (a).

During the IF-FMEA analysis of the task and as we systematically examine the task output (o) for potential failure modes, we will have to consider the possibility of the output being *stuck at* a certain value. Part of the examination process is to identify deviations of the input (m) that can cause the *stuck at* failure at the output. An obvious case of such a deviation is the *omission* of input. For as long there is an omission of input, the output will be stuck at a value defined by the average of the k last valid measurements. More importantly the *stuck at* failure may persist following the end of a temporary omission. Indeed, *if the omission is long enough* to create a deviation between the restored measurement and the last valid average which is greater than ε, *then all new measurements will be discarded as invalid*, i.e. we will have a persistent stuck at failure.

Let us now assume that the task is part of the wheel node implementation. The task input is the pedal message arriving through the bus and the task output is the braking pressure applied to the wheel. Our analysis has shown that if there is a *temporary omission* of the pedal message at the early stages of braking (e.g. due to electromagnetic interference), the output might be *permanently stuck at zero or at a low braking value* which will cause *a failure to brake*. This problem was rectified in a redesign of the peak detection algorithm.

7.4. Fault tree synthesis

In the course of this study we also mechanically generated, regenerated and evaluated a number of fault trees for the brake-by-wire system. Fig. 19 illustrates, for example, the fault tree that we synthesised for one of the main hazardous functional failure modes of the system: “omission of braking”. We must clarify that the intention of this figure is to illustrate the level of complexity and the form of the mechanically generated trees rather than to show details of the case study.

The mechanical analysis of the fault trees that we synthesised pointed out weak areas of the design and focused our efforts on those areas. The *minimal cut-set* analysis, for example, pointed out single points of failure and areas of the design that contributed more to the overall failure probability of the system. These results initiated a number of useful design iterations and guided the revision of the

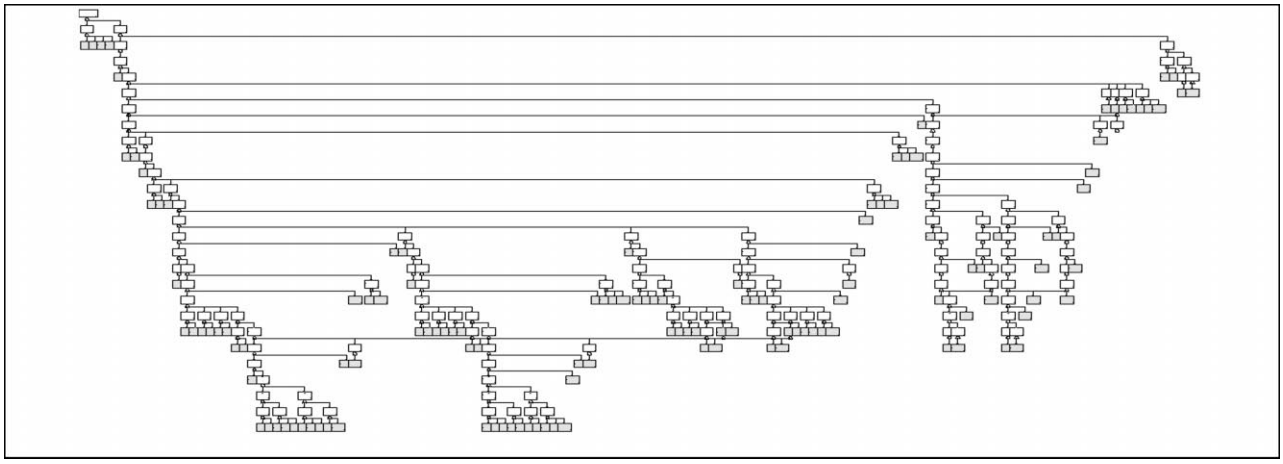


Fig. 19. Distant view of the fault tree for the event “Omission of braking”.

fault tolerance strategies in the system and the allocation of additional redundant resources.

It is equally important to point out that the synthesis algorithm could not have generated those fault trees if there were any inconsistencies in the hierarchical model or among the analyses of the components of that model. In that case, the algorithm would have simply pointed out the inconsistencies. The synthesised fault trees, therefore, link in a consistent manner the results from the various analyses to each other and back to the high-level FFA, and hence guarantee the consistency of the safety case.

8. Conclusions and further work

In this paper we identified a number of problems arising in complex safety assessments and proposed a way to address those problems by extending, automating and integrating a number of classical techniques into a new method for safety analysis.

The proposed method enables the assessment of a complex system from the functional level through to low levels of the hardware and software implementation of the system. It integrates design and safety analysis, and in the process of the assessment, links a consistent hierarchical model of the system to the results from the safety studies. The method also harmonises hardware safety analysis with the hazard analysis of software architectures, and introduces an algorithm for the synthesis of fault trees, which mechanises and simplifies a large and difficult part of the analysis, the development of fault trees.[7]

We described the method and demonstrated its application on a distributed brake-by-wire prototype in a laboratory environment. In the course of our presentation, we attempted to address two questions concerning our approach to safety analysis. Firstly, can the method help us rationalise and simplify safety assessment, and generate consistent safety cases? Secondly, can the results from the analysis help us improve the failure detection and recovery mechanisms of the system under examination and, if so, how? We believe that the brake-by-wire case study demonstrates positive results with regard to both those questions.

Firstly, the fault trees that we mechanically generated guarantee consistency among the low-level safety analyses and between those analyses and the hierarchical model. They also indicate that HiP-HOPS can rationalise the development and maintenance of large fault trees, and, in that sense, can alleviate some of the problems currently encountered in the quantitative aspects of complex safety assessments.

Secondly, the results from the analysis helped us to improve systematically the failure detection and control mechanisms of the brake-by-wire system. The FFA, for example, helped the design of mechanisms for the recovery from single and multiple wheel locking failures. Sensor IF-FMEAs directly supported the design of hardware redundancy schemes and averaging or voting algorithms for the detection and masking of sensor failures. IF-FMEAs also helped us analyse the pedal and wheel node architectures, and to identify subtle errors in the design of certain software algorithms. Finally, the mechanical analysis of fault trees further helped us to identify weak areas of the design and focus our efforts to those areas.

Our limited experience from the application of the method indicates that the method can rationalise and simplify an inherently difficult and costly task, the safety assessment of programmable electronic systems. The method, though, also inherits some of the limitations of classical safety analysis. In the quantitative aspects of the assessment, for example, we still rely on component failure rates (λ) which are often difficult to obtain and the validity and value of which are generally disputed [19,20].

A second limitation of HiP-HOPS is that the predominantly structural model currently providing the basis for the analysis does not enable the representation of highly interactive systems where operator errors can contribute significantly to the failure of the system. The application of the method is therefore currently restricted to electromechanical systems that have limited interaction with human operators. Finally, we must point out that the process that we described in this paper requires that the *behaviour or configuration of the system remains stable within the period of operation covered by the analysis*. If the system undergoes behavioural or structural transformations during operation, a separate set of analyses must be carried out within each phase of operation.

We currently extend our method to resolve the complications caused in safety analysis by the often interactive or dynamic character of complex systems. Our approach is to introduce in the centre stage of the assessment process a dynamic model that can capture the behavioural of structural transformations that occur in such systems. In its general form, this model is a hierarchy of abstract state-machines, which is structured around the current static (structural) hierarchy of HiP-HOPS. We hope that we will soon be in a position to demonstrate that, with those extensions, the principles that we have developed in this paper could also be applied effectively in the context of highly dynamic or interactive systems.

Acknowledgements

The authors would like to thank the European Commission for the continuing support of this research.

[7] We are aware of other approaches to the synthesis of fault trees, which exhibit some conceptual similarity to our approach, in that they also use a description of the system to trace dependencies and the propagation of failure between components (see for example, [21–24]). However, it is beyond the scope of this paper to discuss the precise relationship of this aspect of HiP-HOPS to previous attempts for the automatic synthesis of fault trees.

References

[1] Society of Automotive Engineers, ARP-4761: Aerospace recommended practice: guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, 12th edition, SAE, 400 Commonwealth Drive Warrendale PA United States, 1996.

[2] Kletz T. HAZOP and HAZAN: Identifying and assessing process industry standards. 3rd ed. Washington, DC: Hemisphere, 1992 (ISBN: 1-56032-276-4).

[3] Palady P. Failure modes and effects analysis. PT Publications, West Palm Beach, FL, 1995 (ISBN: 0-94545-617-4).

[4] Vesely WE. Fault tree handbook, US Nuclear Regulatory Committee Report NUREG-0492, US NRC, Washington DC, United States, 1981. p. X.15–8.

[5] Heiner G, Thurner T. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In: Proceedings of FTCS-28, June 1998. p. 402–7.

[6] Kopetz H, Damm A, Koza C, Mulazzani M, Schwabl W, Senft C, Zainlinger R. Distributed fault tolerant real-time systems: the MARS approach. IEEE Micro 1989;9(1):25–40.

[7] Kopetz H, Grünsteidl G. TTP: a protocol for fault tolerant real-time systems. IEEE Computer 1994;27(1):14–23.

[8] McDermid JA. Support for safety cases and safety arguments using SAM. Reliability Engineering and System Safety 1994;43:111–27.

[9] Yourdon E, Constantine L. Structured design: fundamentals of a discipline of computer program and systems design. Englewood Cliffs, NJ: Prentice-Hall, 1986 ISBN: 0-13854-471-9.

[10] Budgen D. Combining MASCOT with Modula-2 to aid the engineering of real-time systems. Software Practice and Engineering 1985;15(8):767–93.

[11] Kopetz H. The time-triggered approach to real-time system design. In: Randel B, Laprie J-C, Kopetz H, Littlewood B, editors. Predictably dependable computing systems, ESPRIT Basic Research SeriesBerlin: Springer, 1995 (ISBN: 3-54059-334-9).

[12] Ezhilchelvan PD, Shrivastava SK. A characterisation of faults in systems. In: Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems, LA United States, 1986. p. 215–22.

[13] Bondavalli A, Simoncini L. Failure classification with respect to detection. In: Predictably Dependable Computing Systems — First year Report, Task B, vol. 2, May 1990.

[14] McDermid JA, Pumfrey DJ. A development of hazard analysis to aid software design, COMPASS'94, Gaithersburg MD. Silver Spring, MD: IEEE Computer Society Press, 1994.

[15] Fenelon P, McDermid JA, Nicholson M, Pumfrey DJ. Towards Integrated Safety Analysis and Design. ACM Applied Computing Review 1994;2(1):21–32.

[16] Fenelon P, Kelly TP, McDermid JA. Safety cases for software application reuse. In: Proceedings of the 14th International Conference on Software Safety, Reliability and Security (SAFECOMP '95), Belgirate, Italy, 1995.

[17] Aho AV, Sethi R, Ullman JD. Compilers: principles techniques and tools. Reading, MA: Addison-Wesley, 1986 ISBN: 0-201-10194-7.

[18] International Electrotechnical Commission 65A/179-185, IEC-61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, IEC, 3 rue de Varembé CH 1211 Geneva Switzerland, 1997.

[19] O'Connor PDT. Fundamental limitations of reliability prediction and modelling. In: O'Connor PDT, editor. Practical reliability engineering, New York: Willey, 1991. p. 111–7 (ISBN: 0-47192-696-5).

[20] Khrishna CM, Shin KG. Reliability evaluation techniques. In: Khrishna CM, Shin KG, editors. Real-time systems, New York: McGraw Hill, 1997. p. 327–60 (ISBN: 0-07-0557043-4).

[21] Salem SL, Apostolakis GE, Okrent D. A new methodology for computer aided construction of fault trees. Annals of Nuclear Energy 1977;41:417–33.

[22] Apostolakis GE. CAT: a computer code for the automated construction of fault trees. EPRI Technical Report NP-705, Electric Power Research Institute, Palo Alto California United States, 1978.

[23] Taylor JR. An algorithm for fault tree construction. IEEE Transactions on Reliability 1982;R-29(1):2–9.

[24] Poucet A. STARS: Knowledge Based Tools for Safety and Reliability Analysis. Reliability Engineering and System Safety 1990;30:379–97.