Dependable Risk Analysis for Systems with E/E/PE Components: Two Case Studies

Jörn Stuphorn, Bernd Sieker and Peter B. Ladkin

Causalis Limited

Bielefeld, Germany

Abstract

1 Introduction

There is a major question of how to perform an accurate risk analysis of systems with software-based components (often subsumed under the rubric electrical/electronic/programmable-electronic, or E/E/PE systems). There is a consensus amongst senior scientists and engineers, backed by rigorous statistical reasoning, that developing systems by 'the usual methods' and testing to identify and eliminate faults cannot attain the required dependability. Other methods are needed, and again the consensus is that these methods must be rigorous, which means formal. It is important that

- the methods connect with the usual methods used by system safety engineers, and
- that they admit practical application to typical industrial examples.

Computer scientists have many formal methods at their disposal whose capabilities are well-known, but which methods are not typically used in industrial development, for various reasons, amongst them that they violate one of these two conditions. We relate in this paper two case studies of how a particular approach, Ontological Hazard Analysis (OHA, first proposed in under the name 'Ontological Analysis' (Ladkin 2005)) can be used for risk assessment of E/E/PE systems.

The basis for OHA is to start with a very abstract requirements specification, of a form which computer scientists are used to produce, in a semi-formally-controlled language. This initial language L must be such that

- it is a formal language, containing names for objects, and symbols for properties of those objects and relations between them, i.e., it is a subset of the language of predicate logic

- the set of all possible (non-equivalent) statements in the language is *finite*

- all users can agree on which of these statements state safety requirements, say the set S

- the safety requirements identified can be seen to constitute a sufficient set

There is some skill involved in picking this first language, and the success of the OHA method is dependent on a suitable choice. The finite set of non-equivalent statements in L must also be small enough that engineers can consider them all, and make judgements about them, in a reasonable period of time.

OHA proceeds from L and S by *formal refinement*, a well-known technique in computer science but not one common amongst system safety engineers. The language L is extended, by considering new objects, properties and relations which express structure and behavior of the system in more detail, less abstractly. Let us call this language L1. The concepts of L (objects, and especially properties and relations) must be expressed in L1. The definitions of the concepts are known as *'meaning postulates'*. The safety requirements in S have their translations into L1, producing say the set of requirements S1, and these are the safety requirements that have to be assured. It may be necessary to introduce new requirements in L1 that guarantee (logically imply) the requirements in S1. Thus the set of safety requirements in L1 is a set S1' which includes S1. This process is repeated as many times as it takes to achieve the goals, which may be

- a system architecture, at, for example a source-code level, so that code may be developed directly from the architecture

- a system architecture which allows standard methods of risk analysis to be applied

We call the successive languages *levels.* The initial language L is *Level 0*, its successor L1 *Level 1*, and so on.

The important feature of the refinement process is the traceability it enables between initial, very abstract system functional definition and, in the end if all goes well, the source-code-level design. This traceability eliminates much of the uncertainty in the development process which leads to unreliability of the risk assessment of the resulting system.

Good idea, but does it work? Many formal approaches do not pan out when applied to industrial examples. We have performed three OHAs on industrial examples. The three analyses were all very different in both style and formal techniques used, but they were all successful in reducing risk assessment to generic methods, and all used the same semi-controlled language/controlled refinement approach of OHA.

1. The first author defined a generic communications-bus architecture applicable to both CAN-bus and Flexray-based communications for road vehicles. The initial language in which the functional requirements were stated was moderately complex. The refinements were achieved through applying HAZOP to the current level, then performing a partial causal analysis of how these deviations could occur (per deviation a mini-Why-Because-Graph, called an epWBG, was created) and the vocabulary necessary for expressing these causal factors defined the next level. The analysis was moderately complex, as he says. However, the epWBGs could be easily converted into fault-tree representations, and already at Level 2 the separate mini-fault trees resulting from the epWBGs could be combined into a single fault tree, enabling the usual fault-tree risk-analysis method of assigning probabilities to the leaf nodes and working one's way upwards through the tree. Thus the goal was accomplished of taking a moderately-complex and realistic E/E/PE system and developing it to the point at which state-of-the-practice risk analysis methods could be applied. Any residual unreliability of such an analysis resides in the usual difficulties with fault-tree analysis (the accuracy of the necessary probabilistic-independence assumptions, for example) as well as in the confidence of the accuracy of the derivation of the fault tree. (We admit a certain amount of laziness here – the actual derivation of the fault tree was performed as a student project at the University of Bielefeld, where the third author teaches. Thus we confirmed that the conversion is feasible, which was the point of the exercise, but we did not necessarily arrive at a fault tree which we would trust!)

2. The second author attempted to derive a computer-based system for performing the communications between train controller and drivers necessary for operating trains according to the German train-dispatching protocol for non-state-owned railways. Train dispatching (German 'Zugleitbetrieb') is the common means of operating trains on single-track lightly-used rail lines, which are commonly not equipped with signalling systems. The protocol is defined in a document, the FV-NE, which is part of German administrative law. He started from the obvious, overriding requirement for block-based train protection, that no two different trains may occupy the same block at the same time except under certain special circumstances. The Level 0 language required to express this is astonishingly simple, and enabled a manual selection of safety requirements, which is complete in the sense that they cannot be logically strengthened. Level 1 and further levels were defined through the usual type of refinement process familiar to computer scientists, in which the extensions of the language were carefully controlled in small steps. It proved to be possible to express the entire functional operation of the system at each level in terms of a global finite-state machine, and the state machines were formally proved to refine each other, sometimes through addition of extra requirements which then become safety re-

quirements. The final step involved transforming the global state machine into a set of communicating state machines, one representing a driver and one a train controller, with message-passing. This was expressed in a structure called a Message Flow Graph (MFG), for which the third author has defined a formal semantics (Ladkin and Leue 1995), and thus the MFG could be formally proved to implement the appropriate global state machine. The MFG agents were then implemented as SPARK procedure skeletons with the appropriate annotations by Phil Thornley of SparkSure, and the annotation proved to implement the MFG. Thus the entire development ensured complete traceability between very-high-level safety requirements and SPARK source code. Suppose such a system were to be implemented as either an automated dispatching system, with computers replacing the human agents, or, more practically, as a support system which checks that the required steps have been performed by the human agents. Then the risk of using the system resides entirely in the hardware and communications systems used, as well as in the compiler used to compile the source code, and in human factors such as whether the system is used as intended, and there is no residual risk inherent in the logic of the program design itself. The risk of this computer-based system has thereby been reduced to that of other, generic risks, which data from other, unrelated projects may be used to assess.

3. The first two authors have performed a security analysis for a configuration control and downloading system for road vehicles with configurable components based on generic hardware, in the European Commission Integrated Project AC/DC, which involves a number of large European automobile and component manufacturers. The secure downloading of a configuration from secure manufacturer sources to a vehicle in the field is a vital component in the process which the project is attempting to define and prototype. The authors first defined a threat model, with which their project clients agreed, and then using OHA derived a complete set of attack patterns and therefrom the attack trees for this threat model. No other technique is known to us which could have accomplished this in a checkably-reliable way. The total effort involved was eighteen person-months, a non-trivial amount but still a low level of effort when compared with the consequences of a successful attack. Since this example concerns security and not safety, we do not consider it further here.

Conclusion. The field of E/E/PE safety lacks methods for performing risk analysis on systems with software-based components in such a way that one may be confident in the risk assessment. The technique OHA, based on expression of requirements in semi-controlled language and formal refinement steps, allows the risk assessment of an E/E/PE system to be based on generic state-of-the-practice risk-assessment methods, in such a way that one may be as confident in the results of an assessment as one is confident in these generic methods. The application of OHA may be straightforward or more complex, but in our case studies on industrial examples it has lain within the range of the economically achievable. We thus recommend its use.

Structure of the Paper. We have stated above the purpose and conclusions, as well as briefly described the case studies concerning the use, of Ontological Hazard Analysis. This constitutes, if you like, the ‘executive summary’ of the work. The two following sections present some details of the first two case studies

2 First Case Study: OHA for an Automotive Communications Bus System

Bus communication systems in road vehicles became useful with the integration of increasing numbers of electronic devices. The multiplexing of these at first separated systems via a communications system enabled savings in weight, lower costs of production, and greater design flexibility.

With emerging new areas of application such as X-by-Wire, communication protocols supporting time-triggered communication are an increasingly common sight in cars.

2.1 Initial System Description

Schematically, an integrated communication bus system in a car can be depicted as shown in Figure 1. The operator of the vehicle gives input into the system using steering wheel, pedals, shift box and other selector switches, of which the states are assessed by sensors which provide input for network Nodes (NIC). These are interconnected with a network bus by which information exchange is enabled. Other Nodes process the available information and provide them to connected actuators with can then influence brakes, gear, inverter, transmission, etc.

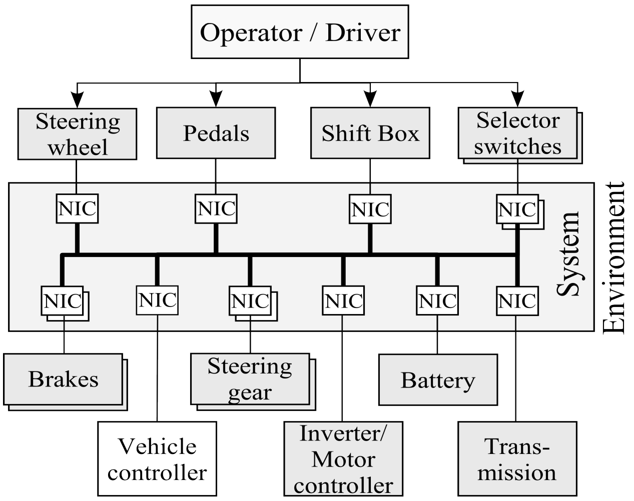


Fig. 1. Integrated Communication Bus System

For the identification of hazards to the communication bus, the system is defined to compass the nodes and the physical wiring of the network bus; all other elements are part of the environment.

2.2 Ontology of the initial system description

Based on the initial system-description three objects with ten properties and one relation are identified. To avoid misunderstanding the meaning of each element of the ontology is defined with the element in the tables below.

Table 1. Objects of the System

Object	Description
NIC	The Network Interface Controller. This is the interface between the input device and the physical network.
Wiring	The physical connection between the systems' NICs. Transmission
Transmission	The transport of information between NICs over the physical network.

Table 2.a. Properties of NIC

Property	Description
Input	The information received by the NIC
Output	The information transmitted by the NIC
Intact	The integrity of the NIC, whose absence prevents the NIC from working properly.

Table 2.b. Properties of Wiring

Property	Description
Intact	The integrity of the wiring, whose absence prevents the physical network from working properly.

Table 2.c. Properties of Transmission

Object	Description
Size	The size of the transmission
Deadline	The latest possible point in time at which the transmission can be received without loosing its value.
Period	Frequency of the generation of a type of transmission
Mode	The mode used for a transmission. This can be either time-triggered or event-triggered.
Latency	The time it takes for the complete transmission of information over the network.
Jitter	The variance in the transmission time of a multitude of same-typed transmissions.

Table 3. Relations of the System

Relation	Description
Connection(Wiring, NIC)	The feature of the NIC to be connected properly with the Wiring.

2.3 Guide-Word based Approach for Identification of Hazards

We used HAZOP's guide-word-based approach to identify deviation because of its systematic nature.

By combining the HAZOP guide-words with each element of the ontology, a comprehensive list of possible deviations is generated. As usual in HAZOP, these possible deviations now have to be interpreted for their impact and meaning in the specific application. A number of these putative deviations can easily be dismissed, as certain guide-words may not make sense when applied to certain elements.

Table 4 HAZOP guide-words used and their interpretations

Guide-Word	Source	Interpretation
No	RSC01	None of the design intent is achieved
	RCC99	This is the complete negation of the design intention - No part of the intention is achieved but nothing else happens
More	RSC01	Quantitative increase in a parameter
	RCC99	This is a quantitative increase
Less	RSC01	Quantitative decrease in a parameter
	RCC99	This is a quantitative decrease

As well as	RSC01	An additional activity occurs
	RCC99	This is a qualitative increase, where all the design intention is achieved together with additional activity
Part of	RSC01	Only some of the design intention is achieved
	RCC99	This is a qualitative decrease, where only part of the design intention is achieved
Reverse	RSC01	Logical opposite of the design intention occurs
	RCC99	This is the logical opposite of the intention
Other than	RSC01	Complete substitution. Another activity takes place
	RCC99	This is a complete substitution, where no part of the original intention is achieved but something quite different happens
Early	RSC01	The timing different from the intention
	RCC99	Something happens earlier in time than intended
Late	RSC01	The timing different from the intention
	RCC99	Something happens later in time than intended
Before	RSC01	The step (or some part of it) is effected out of sequence
	RCC99	Something happens earlier in a sequence than intended
After	RSC01	The step (or some part of it) is effected out of sequence
	RCC99	Something happens later in a sequence than intended
Faster	RSC01	The step is done with the right timing
Slower	RSC01	The step is not done with the right timing
Where else	RSC01	Applicable for flows, transfers, sources and destinations

The list of guide-words shown in Table 4 is a combination of guide-words proposed by the Royal Society of Chemistry (Hazell et al. 2001) and (Redmill et al. 1999).

Overall our system ontology for the initial system description comprises 14 elements and the set of guide-words 13 elements. The combination of elements with guide-words produced 182 possible deviations which were reduced by the interpretation process down to 59 meaningful deviations, a reduction of about 67%.

2.4 Formalisation of Deviations by Usage of Ontology

The systematic generation of deviations produces some equivalent deviations in varying wording. Such deviations do not have to be analysed more than once, but can be difficult to identify. We accomplished this by expressing the deviations semi-formally using the vocabulary of the ontology. E.g. the deviation *'Information is reversely transmitted'* can be expressed by the formula *'Output(NIC) = INVERSE(Input(NIC))'*. Equivalences are much easier to see using the semi-formal mathematical-style language.

As a side effect, this formalisation helps to identify missing elements in the ontology, which can then be included to enable the expression of further deviations.

In the step from the initial system description and ontology to the first refined version, this led to an additional 3 objects, 21 properties and 1 relation. The refinement to the second refined version identified another 14 properties and 1 relation.

After three iterations of refinement the system ontology overall comprises 6 objects, 45 properties and 3 relations.

2.5 Extended Partial Why-Because Graphs

To analyse the causal factors leading to a deviation, an extended partial Why-Because Graph (epWBG) is created. Why-Because Graphs were intended for a-posteriori analysis of incidents, in which all causes of a node actually occurred (Ladkin 2000). We could say by analogy with fault trees, that the graph-relationships are all AND-related. For system development, we need to consider alternative ways in which an event can occur, and thus one needs to represent an OR-type relationship as well, as in e.g. Mackie's INUS conditions (Mackie 1974). The WBG is extended by introducing an OR relationship, and because we are only concerned with limited causal relationships among certain elements, we call the result an extended partial WBG or epWBG.

Typically the epWBG describing the causes of the occurrence of a deviation are rather small, the number of their nodes varying between 1 and 11. For example, the events that can cause the deviation *'The Network has no shielding'* which can be expressed as *"Shielding(Network) = 0"* to occur can be represented as in Figure 2.

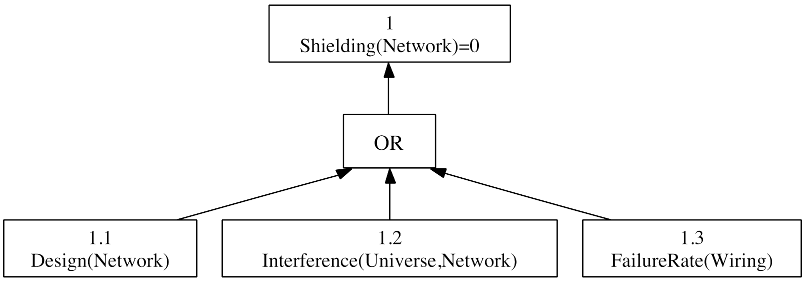


Fig. 2. epWBA of deviation 'The Network has no Shielding'

From the system definition only three events can lead to the deviation occurring: either the shielding was omitted during design; direct interference from outside the system caused the shielding to disappear; or the shielding failed by itself.

Other deviations are more complex in their causal description. The causes of the event of a network node becoming dysfunctional or broken, *'NOT Intact(NIC)'*, are shown by the epWBG in Figure 3.

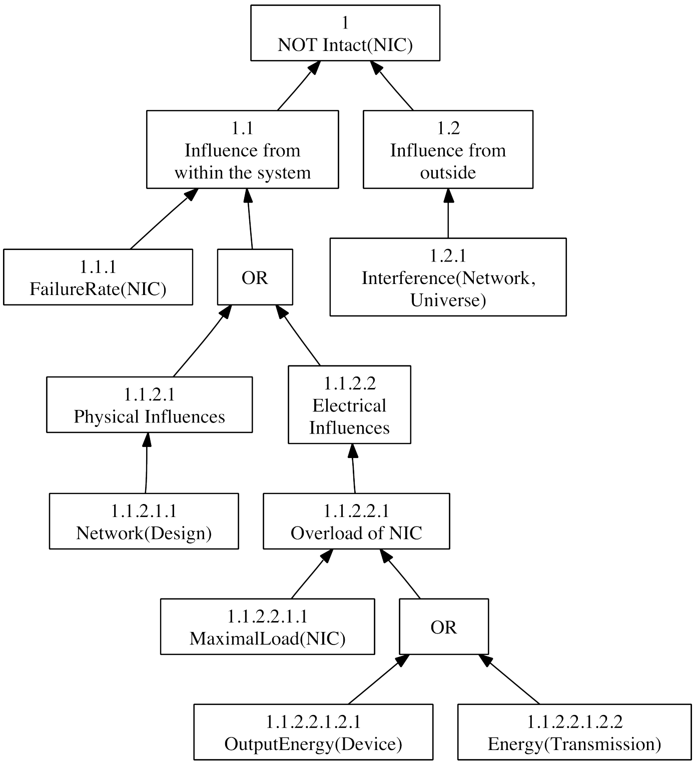


Fig. 3. NIC is not intact

2.6 Statistics of the Analysis

As shown in Figure 4, the elements in the ontology of the system description expanded most in the first refinement step. The step from 2nd to 3rd iteration also provided a more detailed system description; the missing elements were mostly properties of objects and one relation.

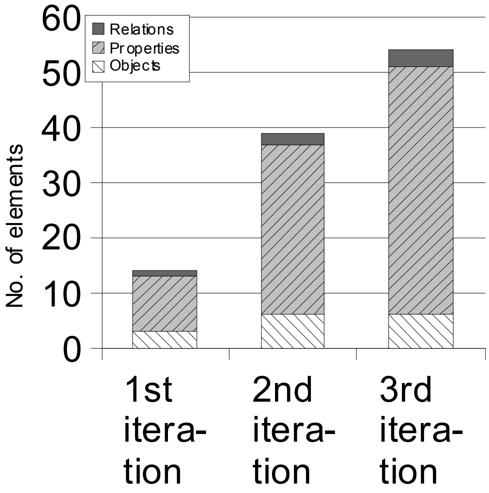


Fig. 4. Extent of Elements in System Description's Ontology

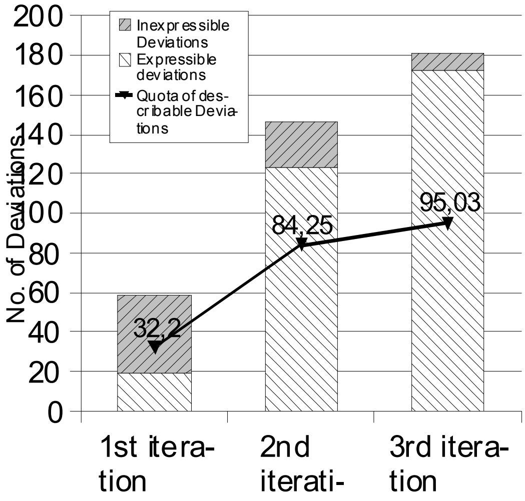


Fig. 5. Expressible and Inexpressible Deviations

In Figure 5 the overall numbers of deviations are shown, classified into deviations expressible with the system description's current ontology and those so inexpressible. The refinement step is expressly intended to be able to state these deviations. As can be seen, with advancing refinement of the system description, the percentage of expressible deviations continually improves.

2.7 Transformation of epWBGs into Fault Trees

For risk assessment of the system, it is necessary to quantify the possible failures. One common way to do this is through a fault-tree analysis. A fault tree was created by first translating the epWBGs into corresponding small fault trees, which were then combined into an overall fault tree describing all the possible factors leading to a failure. This transformation was performed by a group who were learning how to work with fault trees. The goal was not to produce a fault tree suitable for troubleshooting and system maintenance, which requires that nodes adjacent to the root-node act as decision points, but rather to produce a fault tree which could be used for risk assessment, in which leaf nodes are assigned probabilities and the probabilities are combined moving up the tree towards the root-node. Thus, when constructing the combined fault tree, certain 'classification nodes' were introduced to denote clusters of similar factors without regard as to whether these classifiers were observable. So e.g. *Human failure* was used as such a classifier and would obviously not be appropriate in a fault tree used for diagnosis.

2.7.1 Filtering of epWBGs

During the course of the analysis, several epWBGs were built which identified problems residing in the specification. As the goal of the fault tree lies in the assessment of risk for an implemented system, such specification faults were not included in the combined fault tree, for they would be eliminated before the implementation stage.

Another feature of the deviation-identification approach is the identification of trivial events such as *'The device does not exist'*. In most cases, such events occur also through failures in specification or the implementation and would similarly be eliminated before the implementation stage and were not included in the combined fault tree.

epWBGs comprising only two nodes resolve to an identity in fault-tree notation. They occur as one node in the generated fault tree.

2.7.2 Algorithm used for clustering epWBGs

As the epWBGs are formulated to describe deviations, one epWBG can describe factors involved in other epWBGs. To cluster these, the following procedure was used:

1. Choose one epWBG

2. Look at leaves

3. Select concepts in leave nodes

4. Look up concepts in HAZOP tables

5. Identify the interpretation that fits the node in HAZOP table

6. Go to the list of identified deviations and identify the respective deviation number

7. Repeat process for the epWBG for the identified deviation

The application of this procedure led to several combined epWBGs which formed the basis for the next step, the transformation into one larger fault tree.

2.7.3 Conversion of clustered epWBGs into partial Fault Trees

A typical example for the conversion from an epWBG into a partial fault tree is shown below and should be self-explanatory given the above comments.

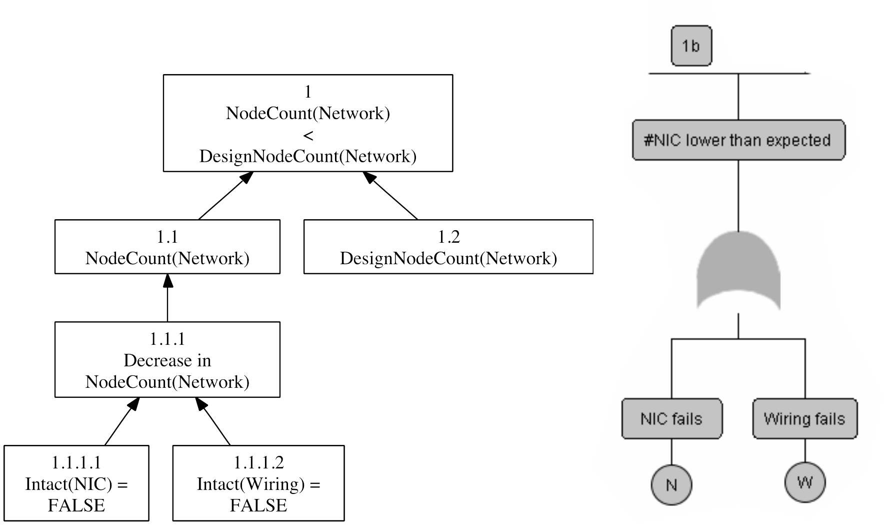


Fig. 6. epWBG formulated to describe deviation and the resulting Fault Tree

2.7.4 Combining partial Fault Trees into one overall Fault Tree

As root node for the fault tree the event *'Problem occurs'* was chosen, a nondescript, but generic name for all system failures identified in the OHA.

Investigation of the epWBGs revealed that all failures could be classified under the topics *'Human failure'*, *'Information not transceived'* and *'No data from device'*. The resulting head of the Fault Tree is shown in Figure 7.

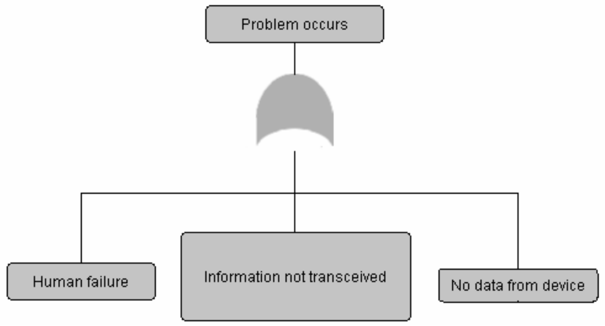


Fig. 7. Head of Fault Tree

Then the partial fault trees were sorted according to their respective classification. The resulting fault tree comprises about 150 nodes. This is of a size often encountered in industrial fault tree analyses and the risk calculation can be handled by the usual methods. The fault tree represents only a certain level of refinement of the system, however, this refinement suffices to allow an arguably realistic assessment of risk given the usual probabilistic independence assumptions in fault tree analysis. We would caution however, that such independence assumptions must themselves be carefully analysed in order to ensure they hold. Our analysis did not go this far.

3 Second Case Study: OHA of Train Dispatching

This work formalises the German train-dispatching protocol for non-state-owned railways ('Zugleitbetrieb'). Administrative law (VDV 2004) sets the requirements for how this is to be done. We derive a system expressed in SPARK source code which implements a (completed version of) this legal protocol.

Complete traceability is maintained between the abstract high-level safety requirements and the SPARK source code through formal refinement. Were the SPARK code to be implemented in communicating machines which either back up or replace the human agents of the system, then the risk analysis of the system may assume that the logic of the communications is faultless. The residual risk consists of the risks associated with the ADA compiler, the hardware used for running the code and for the communications, and human factors.

A set of safety requirements which are guaranteed to be adequate are derived by starting with a very simplistic, seemingly trivial description. The safety requirements are determined for this first level (Level 0) by enumerating all possible truth functions for two trains in the available language, and determining which of these are safety requirements.

The original Zugleitbetrieb (ZLB) relies on a single human operator (the dispatcher, or Zugleiter) to make sure that a given track section is free before allowing any train to enter that section. There are no signals and other supporting technology to locate trains. The system, as well as its derived system developed here, relies solely on messages passed between the train conductors and the dispatcher.

2.2 Ontological Hazard Analysis

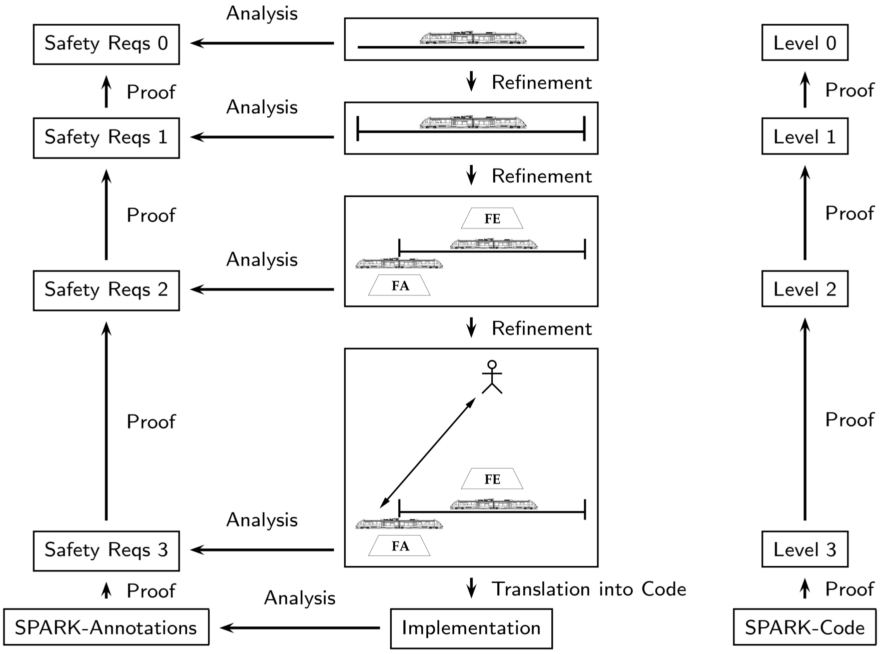


Fig. 8. Structure of the OHA

3.2.1 Starting the OHA --- Level 0

The goal of the highest specification level, Level 0 is not to provide a detailed description of train operations, but to provide a description that is so simple that we can define safety axioms to which all applications experts can assent and at the same time ascertain that these axioms are both *correct* and *complete* relative to the expressions of the language.

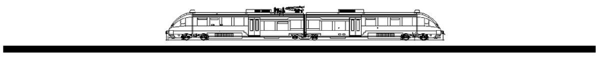


Fig. 9. Schematic Representation of Level 0

Table 5. Level 0 Sorts

Sort	Description
Vehicle	Any train or other vehicle operating on tracks
Block	A section of a track inside or outside a station

Table 6. Level 0 Relations

Relation	Description
inA(F,S)	Train F is in Block S
ZV(F,S)	ZV(F,S) Train F may occupy Block S under central responsibility (normal scheduledoperation)
LV(FS)	ZV(F,S) Train F may occupy Block S under local responsibility (special case)

Determining Safety Axioms. Using elementary propositional logic as well some semantic domain knowledge we are able to determine that there turn out to be only 6 safety postulates on Level 0 from consideration of a couple of dozen non-equivalent statements from a total of 256 statements before semantic reduction. We use the following shorthand notation for a train F1 and one block S: LV(F1,S) = LV1, ZV(F1,S) = LZ1, inA(F1,S) = in1; similarly for train F2. The Safety Postulates at Level 0 are shown in Table 7.

Table 7. Safety Postulates at Level 0

Safety Postulate	Description
$ZV1 \Rightarrow \neg LV1$	If a train is in a block under central responsibility it cannot be there under local responsibility
$\neg LV1 \wedge in1 \Rightarrow ZV1$	If a train is in a block and is not there under local responsibility then it is under central responsibility
$in1 \wedge ZV1 \Rightarrow \neg LV1$	If a train is in a block under central responsibility it cannot be in that block under local responsibility
$(F1 \neq F2) \Rightarrow (LV1 \Rightarrow \neg ZV2)$	If a train is in a block under local responsibility another train under central responsibility cannot be in that block
$(F1 \neq F2) \Rightarrow (in1 \Rightarrow \neg ZV2)$	If a train is in a block another train under central responsibility cannot be in that block
$(F1 \neq F2) \Rightarrow (ZV1 \Rightarrow \neg ZV2)$	If a train under central responsibility is in a block, another train under central responsibility cannot be in that block.

3.2.2 Level 1: First Refinement

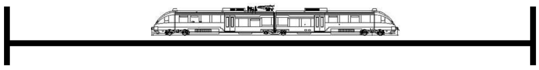


Fig. 10. Schematic Representation of Level 1

The generic block of Level 0 is refined as follows, introducing the new sorts Track and Station. This leads to Table 8.

Table 8. Level 1 Sorts

Sort	Description
Vehicle	Train or other track vehicle
Block	A track section
Track	A piece of track in the station
Station	A station where messages are exchanged

On this level we then have 10 relations. Meaning Postulates define what each Level 0 sort and Level 0 relation means in terms of the Level 1 language.

Using the Meaning Postulates we arrive at 12 Safety Postulates for Level 1.

3.2.3 Level 2

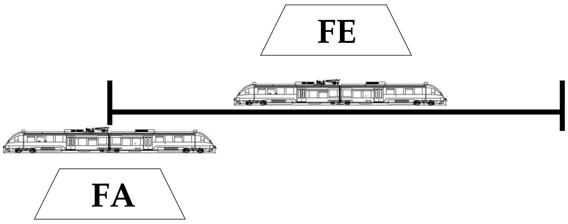


Fig. 11. Schematic Representation of Level 2

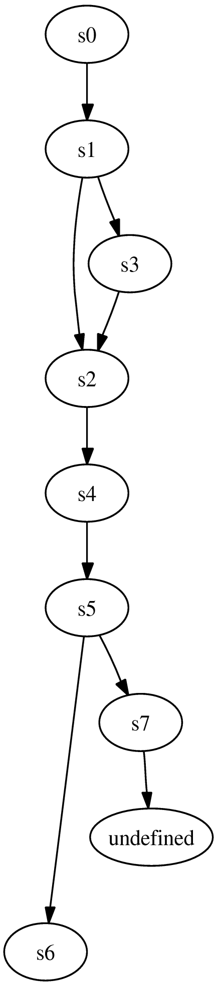
In this level no new sorts are added, but additional relations concerning 'clearances' are added, as shown in Table 9.

Table 9. Level 2 Relations

Relation	Description
FA(F,A,B)	Train F, in station A, has asked for clearance to go to station B
FE(F,A,B)	Train F, in station A, has received clearance to go to station B
AFE(F,A,B)	Train F, in station A, has been denied clearance to go to station B
KH(F,A,B)	No obstructions are known for train F to go from station A to station B

At this point we are now able to build a state-machine representing the global states of clearances which represents a train journey.

The state-machine is shown in Figure 12, which is presented as a Predicate-Action-Diagram (Lamport 1995).



State	Description
s0	inZ(T , A)
s1	∧ inZ(T , A)
	∧ FA(T , A, Next(T , A))
	∧ ¬ FE(T , A, Next(T , A))
s2	∧ inZ(T , A)
	∧ FA(T , A, Next(T , A))
	∧ KH(T , A, Next(T , A))
s3	∧ inZ(T , A)
	∧ FA(T , A, Next(T , A))
	∧ ¬ FE(T , A, Next(T , A))
	∧ ¬ KH(T , A, Next(T , A))
	∧ AFE(T , A, Next(T , A))
s4	∧ inZ(T , A)
	∧ FA(T , A, Next(T , A))
	∧ FE(T , A, Next(T , A))
	∧ KH(T , A, Next(T , A)
s5	∧ zw(T , A, Next(T , A))
	∧ FE(T , A, Next(T , A))
	∧ KH(T , A, Next(T , A))
	∧ ¬ LV(T , S)
s6	inZ(T , A) = s0
s7	∧ btw(T , A, Next(T , A))
	∧ FE(T , A, Next(T , A))
	∧ ¬ KH(T , A, Next(T , A))
	∧ ¬ LV(T , S)

Fig. 12. Level 2 state-machine

Three simple Meaning Postulates and elementary logic leads to only two new Safety Postulates, which can be expressed informally as:

- if no obstructions are known and clearance has been given, the block can be occupied under central responsibility

- clearance for a block cannot be given for a second train, if clearance has already been given for a train for the same block in either direction.

Hazards. The new hazards identified at this level are simply the negations of the newly identified Safety Postulates:

- Clearance has been given, and no obstruction is known, but the conditions for occupying the block under central responsibility have not been met.
- Clearance has been given for two trains for the same block at the same time.

3.2.4 Level 3

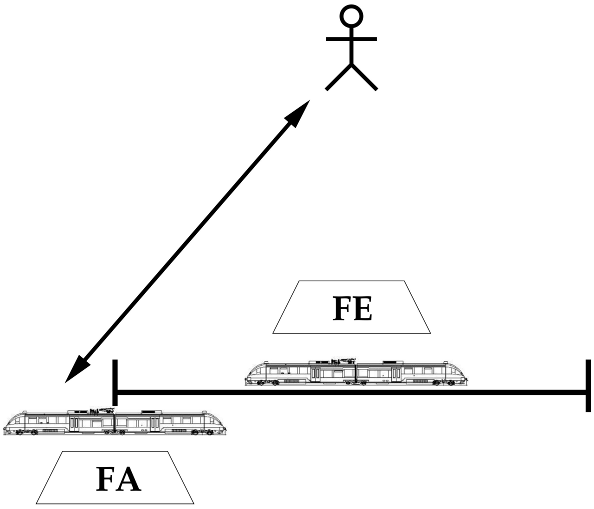


Fig. 13. Schematic Representation of Level 3

Level 3 includes the specific defined communications between trains and a dispatcher.

Message types correspond to the states in which the trains can be, and are designed according to the message types prescribed in the regulations for German non-state-owned railways (VDV 2004).

Table 10. Message types at Level 3

Message Type	Description
FA	Request for Clearance (Fahranfrage)
FE	Clearance (Fahrerlaubnis)
AFE	Denial of Clearance (Ablehung der Fahrerlaubnis)

AM	Notification of Arrival (Ankunftmeldung)

In addition, we define relations to describe sending and receiving of messages, as shown in Table 11.

Table 11. Relations at Level 3

Relation	Description
Sent(MT,T,A)	Message of type MT, concerning train T and station A has been sent.
Recd(MT,T,A)	Message of type MT, concerning train T and station A has been received.

Note that the sender and receiver of the message are implicit. Messages of type FA and AM are always sent by the specific train to the dispatcher, messages of type FE and AFE are always sent by the dispatcher.

Through appropriate Meaning Postulates, the state machine of Level 2 can be augmented to include communications. This now more complex state machine can be transformed into a Message Flow Graph (MFG), to make the communications visually clear. The MFG represents the individual agents and their changing states as vertical lines, message passing between agents as angled lines. The MFG can be formally shown to define the same global state machine as the Predicate-Action-Diagram for this level.

The MFG is used as the starting point to define the SPARK implementation and the SPARK verification conditions are determined by hand to define the MFG of Figure 14.

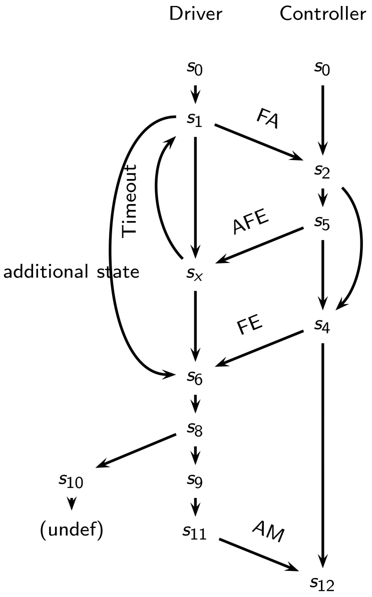


Fig. 14. The Message Flow Graph

Table 12. States corresponding to the Message Flow Graph

MFG-Trans.	Driver-State	Controller State	Global State
s0	inZ(T , A)A)	–	inZ(T , A)
s0 → s1	∧ inZ(T , A) ∧ Sent⟨FA, T , Next(T , A)⟩	--	∧ inZ(T , A) ∧ Sent⟨FA, T , Next(T , A)⟩
s1 → s2	--	Recd⟨FA, T , Next(T , A)⟩	∧ inZ(T , A) ∧ Sent⟨FA, T , Next(T , A)⟩ ∧Recd⟨FA, T , Next(T , A)⟩

3.2.5 The Step to Code: Implementation in SPARK

SPARK is based on a subset of the Ada language. It uses annotations to denote data and information flow and to specify pre- and post-conditions for functions and procedures.

The SPARK tools include a static code analyser that uses the annotations to prove the absence of run-time errors, such as division by zero, buffer overflows and other bounds violations before the code is actually compiled.

SPARK annotations

- strengthen specification
- ‘Design by Contract’
- Allow analysis without access to implementation
- Analysis can be done early, before programs are compilable

SPARK Code Verification Tools

- Examiner
  - Checks control flow and data flow
  - Checks information flow
  - Generates proof obligations (“verification conditions”) for run-time errors
- Simplifier
  - Automatic proof of large majority of proof obligations
  - (Interactive) Proof Checker
  - Used to prove the remaining verification conditions
  - Used to prove conformance of Code to pre/postconditions

Properties of SPARK Code

- Unambiguous
- Bounded space and time requirements
- Free of runtime errors

Code for train dispatching has been completed by Phil Thornley of SparkSure, based on the Message Flow Graphs. Proofs have been completed that the Code fulfils the annotations, and that the annotations fulfil the Level 3 Message Flow Graph description.

Typical Example of SPARK annotations corresponding to the MFG

```
procedure Send_FA (DS : in out Driver_State);
--# global out Messages.Out_Queues;
--# derives Messages.Out_Queues from
--# DS
--# & DS from
--# *;
--# pre D_State(DS) = D_S0;
--# post To_S1(DS~, DS);
```

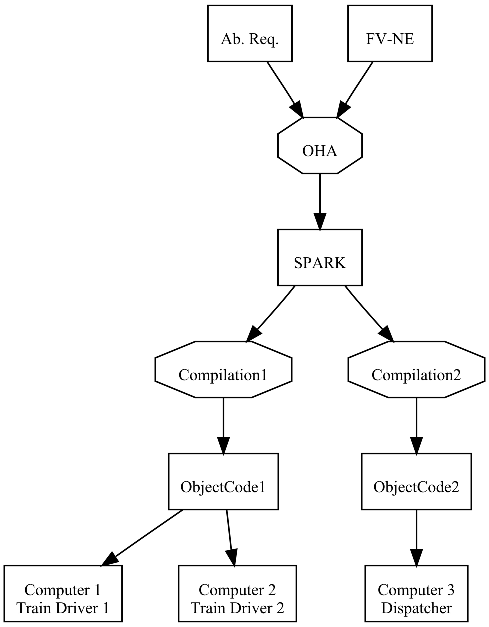


Fig. 15. Summary of Second Case Study

The uninterrupted traceability from Level 0 requirements down to the SPARK source code ensures that the source code fulfils the Safety Requirements of Level 0.

References

Hazell RW, McHattie GV, Wrightson I (2001) Note on Hazard and Operability Studies [HAZOP]. Royal Society of Chemistry, London

Ladkin PB (2000) Causal Reasoning about Aircraft Accidents. In: Koornneef F, van der Meulen M (eds) SAFECOMP 2000. Springer Lecture Notes in Computer Science, 1943:344-360

Ladkin PB (2005) Ontological Analysis. Safety Systems 14(3)

Ladkin PB, Leue S (1995) Interpreting Message Flow Graphs. Formal Aspects of Computing 7:473–509

Lamport L (1995) TLA in Pictures. IEEE Transactions on Software Engineering SE-21

Mackie JL (1974) The Cement of the Universe: A Study of Causation. Oxford University Press

Redmill F, Chudleigh M, Catmur J (1999) System Safety: HAZOP and Software HAZOP. John Wiley & Sons, Chichester

VDV (2004) Fahrdienstvorschrift für Nich-bundeseigene Eisenbahnen (FV-NE). Verband Deutscher Verkehrsunternehmen. Ausgabe 1984, Fassung