CHAPTER 3

Objects, Properties, Relations, Assertions - OPRA

In order to analyse any system and its environment for its safety properties, its hazards and the possible consequences, it is necessary to write things down. What you write down will be descriptions, more or less exact, of how a system is intended to work: what parts it has, how these parts interact, what properties they have and relations to each other, and then eventually to describe what you thing may go wrong or right.

It is important to write these descriptions accurately, and to maintain awareness on what you are emphasising and what you are ignoring. For what you ignore may turn out to be important. In any case, it is a good idea to do about these system descriptions methodically. I give here some examples of how to go about doing that.

3.1 The Pressure Tank

We commence by considering a pressure tank, a canonical example since its appearance in the Fault Tree Handbook [14].

First, we perform the first step of an OHA, identifying the objects, their properties and the relations (OPR) amongst them that will be subject to the hazard analysis at the starting level (reification)

The Pressure Tank

The simple pressure tank is shown in Figure 5.1. It contains three input streams, for steam, hydrocarbon and catalyst, on the left. Each stream is controlled by a valve. The tank itself has a pressure sensor, shown above the tank,

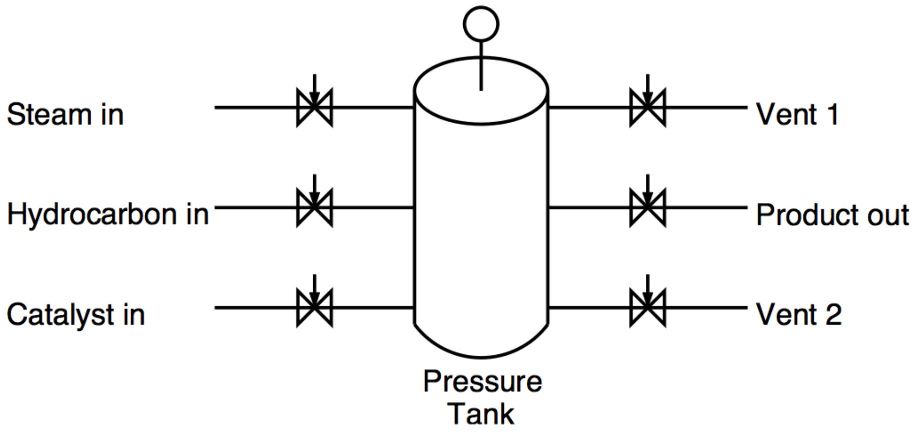


Figure 3.1: The Pressure Tank Without Safety Mechanisms

not currently connected to anything. It contains three output streams, one for the normal output of the product and two vents.

An Accident Informally, an accident is an event which results in possible injury or death to people, or damage to the environment in some way. If a formal definition of terms is required, and at some point it will be, Chapter 9 contains the terms and their explanations which we use here. These differ in some ways from the vocabulary used in the international functional safety standard for electrical, electronic and programmable electronic equipment, IEC 61508:2010 [6], which we shall consider later.

This pressure tank has a steam input line (we can assume: it is drawn as one and contains a symbol for a valve), which could rupture and theoretically cause injury to nearby people; it likely has hot sections (for example an uninsulated steam pipe) which could injure someone who comes into contact with it; maybe very hot sections that could cause other sorts of damage. Pressure tanks are also susceptible to overpressure, which results in rupture that may have a very sudden character, similar to an explosion, resulting not only in physical damage through impact of parts with the surroundings, but also the uncontrolled release of potentially damaging substances into the local atmosphere.

In our later analysis, as an example we shall be concentrating on just the overpressure-rupture event as the dangerous event likely to result in damage - the accident event in our terminology.

The severity of the accident event will depend upon, for example, how many people are in the neighborhood of the tank when it ruptures, and how effluent discharged through the rupture is contained. Most installations will install a pressure tank with possibly damaging contents in some sort of containment structure, as well as restrict the access of people to the containment structure when the tank is in operation. Neither of these mitigation measures are expressible in the preliminary OPRA, so they will not arise in our current analysis, for we do not refine beyond the original OPRA. This is consistent with the way that the other sources consider the pressure tank when constructing the preliminary fault trees. OHA will go further during the refinement process, and other practical hazard analysis methods will also identify the possibility of damage mitigation through containment and restriction of personnel, for these are standard mitigation techniques throughout the process industries. We are concerned here with avoidance of the accident event, not mitigation of its severity.

Preliminary OPRA The design has been given to us by means of a labelled diagram. Given the manifest objects in the diagram, we can specify certain properties and predicates amongst the components of the system through general physical considerations, for example the quantity, temperature and pressure of steam, hydrocarbon, catalyst, and product; the open/closed states of the valves and maybe even which components (tubes, tank, valves) are fulfilling their specification (which is not given here) and which not.

There are other components, such as joints, screws, surface coatings, controlled climate, and so on, which we are not given in the diagram and thus which do not belong to the preliminary OPRA.

At the first stage, we therefore do not assess the state or behavior of these components, although such might be a significant factor in any real accident behavior. For example, the pressure tank may rupture because of high-, not over-pressure, which causes a weak riveted joint in the vessel, that fails to fulfil its specification, to give way. One cannot infer anything about things one is not given, or properties of which one is not made aware. Thus weakness of joints because of non-specification riveting are not part of the preliminary OPRA.

Objects There are quite a few objects to be identified from the diagram, even for so apparently simple an example. We list the names we shall give them and leave the identification of the objects from the names as an exercise for the reader.

- *Tank*
- *SteamPipe*
- *HCPipe*
- *CatalystPipe*
- *ProductOutPipe*
- *VentPipe1*
- *VentPipe2*
- *TankPressureSensor*
- *SteamPipeValve*
- *HCPipeValve*
- *CatalystPipeValve*
- *ProductOutPipeValve*
- *VentPipe1Valve*
- *VentPipe2Valve*
- *Steam*
- *HC*
- *Catalyst*
- *Product*

Properties The following properties pertain to certain objects in this list. They were obtained through considering general physical principles concerning the objects above and their functions, as we did when considering accident events. We again leave the identification of the properties as an exercise.

- *Intact* and its contrary *Ruptured*, to *Tank*, *SteamPipe*, *HCPipe*, *CatalystPipe*, *ProductOutPipe*, *VentPipe1*, *VentPipe2*;
- *Open*, *Closed* and *Partopen*, to *SteamPipeValve HCPipeValve CatalystPipeValve ProductOutPipeValve VentPipe1Valve VentPipe2Valve*;

- *Temperature*, *Pressure*, *Quantity*, to *Steam HC Catalyst Product*. Although we have called these properties, in fact they are what is known as *fluents*, taking values at times, potentially different values at different times.

3.2 OPRA of the Communications Bus

In Chapter 1 I introduced a generic automotive communications bus, pictured again in Figure 3.2. We can begin to list the objects, properties, relations and connecting assertions (here, we shall introduced *meaning postulates*, which declare equivalence of meaning between some of the terms introduced). To obtain the first OPRA collection, I consider how such a bus works, using general information about computer networks and digital transmission devices taken, say from [13], and general knowledge about sensor/controller networks, say from [2].

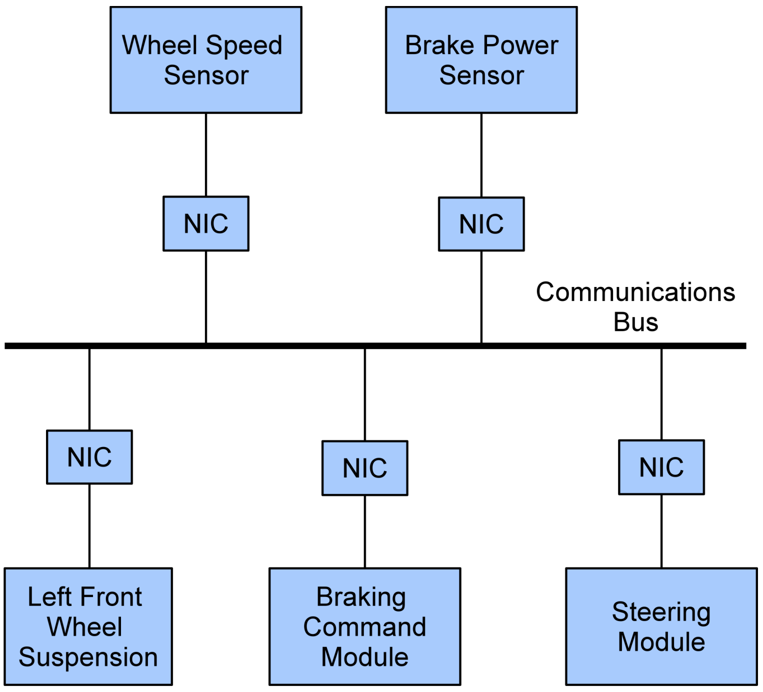


Figure 3.2: The Generic Communications Bus, Again

Sensors measure current values of such phenomena as force at wheels, rotational speed of wheels, position of steering frames, and braking. They produce digital values which are transmitted via the bus to command-units such as steering, motor and brake controllers. These controllers in turn generate commands which are conveyed in digital form to actuators which implement these commands at the steering, motor or brakes. The communication bus is a common medium through which all this information is conveyed.

A single batch of digital information from a single device which is transmitted as a unit will be called a message. At this level we shall be abstracting from the electrical waveform which we anticipate to be present on the bus, and considering its interpretation as a sequence of zeros and ones. There are, however, hazards and failures associated with the interpretation of electrical signals as digital information, such as so-called Byzantine failures [1, 2]. In our considerations here, we shall be ignoring such failures although they obviously have to be taken into account at some point.

Each device, sensor, controller or actuator, is connected to the bus through a Network Interface Controller (NIC). Among the tasks of a NIC is to format the information coming from the sensor in a form appropriate for transmission on the bus using its protocols, for example, adding a unique identifier for the source of the message and its intended recipients. The NIC also coordinates the act of transmitting the message, for example, waiting for an appropriate time slot if the bus is time-triggered; also synchronising its internal clock with those of other NICs so that all NICs have a similar understanding of which times belong to which time slots. Non-time-triggered NICs will also implement a protocol to avoid or reconcile collisions between messages on the bus, which will happen if two NICs try to transmit almost simultaneously.

At this point, one could start to perform a preliminary assessment of the possible failures and dangerous failures of the communication bus. This is often called by system safety engineers a Preliminary Hazard Analysis, or PHA. At this point, though, we do not know very much about the communication bus or the system (the environment) in which it operates. I have said it is for use in road transport, and mentioned brakes, steering and motor, but most of the details of how it is intended to operate are not yet specified.

Elsewhere, we have made much of the difference between a system state (a property of a system) and an event (a change of state, represented best by two states, a before state and an after state) [8]. For the purpose of analysing how an accident can possibly result from system behavior, it will at first be helpful not to distinguish these too carefully: let me call either a state or an event a happenstance and call a happenstance hazardous if the system environment could be such that an accident will inevitably ensue.

For example, a failure of the vehicle to steer left on a command to do so could inevitably result in an accident if the road bends sharply left, and has a thick wall on its right hand side! Similarly, a failure to brake on command inevitably results in an accident if there is a wall right in front of the vehicle. We can call such happenstance, the failure to deliver these commands, as a hazardous happenstance.

We can imagine that, if a command is given to apply the brakes, and that command is not received by the brake actuators, and there is only this single command-path, through messages in the communication bus, between brake command and actuation, and no alternative physical connection, say through a hydraulic-mechanical system, that the brakes will not be applied. I think we can say that this represents a potential danger, that if the brakes do not go on when a driver commands, this is a hazardous happenstance. Similarly, if the steering is only commanded through the communications bus, and not also by an alternative physical connection, and the command to steer left is not received by the steering actuator(s), then we can imagine the vehicle going straight on, and leaving the road, instead of steering left through a bend in the road; again a potential danger. So we may judge that, in these cases, failure to deliver a specific message is a hazardous happenstance.

To continue, we may imagine that if a command to brake harder arrives at the brake actuator and is read as brake more lightly, that this is a hazardous happenstance. Or if a command to steer left arrives at the steering actuator as a command to steer right. So we may judge that, in some cases, the corruption of a message during transmission, or a false reading of its contents by an actuator, could lead directly to an accident and thus can be classified as a hazardous happenstance.

We may conclude that loss of a message may be a hazardous happenstance, and corruption of a message may be a hazardous happenstance, as also may be the incorrect reading of a message by a NIC. These may not be the only hazardous happenstances.

We can't go much more into the details of which kinds of loss are hazardous, and which kinds of corruption, without knowing precisely what is connected to the bus, or the subcomponent specifications. We are considering a generic bus, along with generic NIC specifications, which is going to be used in varied applications by various manufacturers. We know through our considerations at this point only that message loss and message corruption are generic hazardous happenstance.

It may occur to us now to ask what kinds of properties, of what kinds of things (messages), loss and corruption are. We are speaking about information which is conveyed by messages: commands, values and so on. We are speaking, then, of the informational content, the digital interpretation, of whatever passes along the medium as the message is transmitted (and, as mentioned previously, not necessarily of the precise waveform which is actually put on the medium by electronic means by the NICs. We are ignoring here such arbitration difficulties, as when a waveform lies on the boundary of what counts as a one bit and what counts as a zero; furthermore we might know of so-called Byzantine errors, in which, say, a waveform on an arbitration boundary is interpreted as a one bit by one device and a zero bit by another).

3.3 Level 0

An OPRA will progress through the process of formal refinement in OHA. We call these progressions *levels*, and the initial level is Level 0. Level 0 is a very abstract view of the system. As in hierarchical design, a lower level will define objects and properties at a higher level by means of its own objects and properties. For example, in a hierarchical design, a natural number may be defined in terms of a positive-integer number range, and an integer number range in terms of sequences of bits, with operations on bits implementing the mathematical operations on integers. It must then be mathematically proved (verified) that the operations on bits indeed implement the operations on integers at the higher level. Similarly, in a formal refinement it must be verified that the lower level objects, properties and relations indeed implement the higher level objects, properties and relations.

If we were to consider a message as a waveform, its description would be physically quite complicated, and we would have to worry about properties such as attentuation, arbitration boundaries, and so forth. If we are to consider a message as consisting of information in the form of data, ultimately encoded as bits, then its description is much simpler. We must say only what data is to be carried, and not be concerned at this stage with how this is carried out.

So we are considering messages abstractly. Messages may be lost, for whatever reasons: a message is lost if it is sent, but no part received by the receiver. Message-content may be corrupted; that is, the content may be read differently by the receiver than was written by the sender, for whatever reasons. What else could happen with messages at this level of abstraction? Symmetrically to loss, they could be created: that is, a message could be received that was not sent. This may seem somewhat fanciful, until one considers that duplicating a message generates a message that was received (in the second version) but not sent (only the first was sent). Whether messages could be generated in the system, say, by outside electrical influences on the medium, is a matter for more detailed consideration at other stages.

We now define the formal vocabulary at Level 0. That is to say, object types, properties which objects of that type are to have, and relations between objects of those types.

3.3.1 Objects

So far, we have the following types of objects: medium (Bus), NICs, messages (msgs). There is only one Bus, but there are typically many NICs and many msgs. Thus are Bus, NIC, msg types of objects, not names for specific objects (although Bus could be considered as one, since there is just one Bus). We now consider the properties which objects of these types may have.

3.3.2 Properties

Bus. The bus itself transmits messages. This is a relation to messages, though, not a property of the bus itself. A property which the bus has is its integrity: is it doing its transmissive job, or is it broken, cut or otherwise compromised?

We thus have the following property: *Integrity(Bus)*, where we write the object type in the parameter position; it might be preferable to write, similarly to type declarations in programming languages, *Integrity(X: Bus)*.

NIC. The NIC assembles messages from its attached device and transmits the assembled messages to/on the bus. It also receives messages from the bus, disassembles

Objects	Properties
Bus	Integrity
NIC	Integrity
msg	Content Size Deadline

Figure 3.3: Objects and Their Properties

them and transmits relevant information further to its attached device. If it performs this task as it is specified, and (we hope!) thereby designed to do, it retains its integrity. If not, it has lost it.

We have the following property: *Integrity(NIC)*. Again, it might be preferable to write *Integrity(Y: NIC)*, but it seems to me that we can just as well use obvious names such as *NIC1*, *NIC2*, etc, to indicate both an object and its type, at this level of discourse. I shall do so, also with msgs, without being much concerned about the exact syntax.

Msg. As we discussed, a message has content. It also has length, or size. Since the system is real-time, it can be that certain messages (for example, to steer, or to brake) have a deadline by which they must reach their receiver NIC.

We have the following properties: *Content(msg)*, *Size(msg)*, *Deadline(msg)*.

3.3.3 Relations

A NIC may be attached or not to the Bus. We take a NIC to be attached when it receives messages intended for it as receiver, and transmits on the Bus messages which its attached device sends. Similarly a message may be on the Bus, when the waveform corresponding to the message is travelling along the medium. A message may be in a NIC, when it is being assembled or disassembled. It is also sent by a NIC and received by a NIC, during which time it, or rather part of it, is also on the Bus.

We thus have the following relations: Attached(NIC,Bus), On(msg,Bus), In(msg,NIC), Sending(msg,NIC), Receiving(msg,NIC).

The relations may be arrayed in a table as in Figure 3.4, in which an X indicates

Relation	Bus	NIC	msg
Attached	X	X	
On	X		X
In		X	X
Sending		X	X
Receiving		X	X

Figure 3.4: Relations Amongst Types of Objects

that the relation has the object type as an argument:

Figures 3.3 and 3.4, then, constitute the definition of the vocabulary for Level 0 of the communications bus and its analysis.

3.3.4 Meaning Postulates

The property of being *Sent* or being *Received* by a NIC entails that a *msg* is at the same time *On* the bus. Also that it is not yet *In* the NIC. This is part of what we mean by asserting *Sent* or *Received*. It could be otherwise: we could have intended received to mean that a *msg* is in the NIC, having been completely read from the bus by the NIC, and therefore no longer partly *On* the Bus. But we didn't choose this option; we chose the former.

We have to mark this distinction somehow: we have to say what we mean by use of the words. We do this in part through enumerating logical relations between the relations, properties and objects we have thus denoted. We shall call these logical assertions of part-meaning meaning postulates.

We have so far the following partial meaning postulates (they are partial because they do not define an equivalence of meaning, but only state an implication):

$$\mathit{Sending(msg,NIC)} \Rightarrow \mathit{On(msg,Bus)}$$
$$\mathit{Receiving(msg,NIC)} \Rightarrow \mathit{On(msg,Bus)}$$

We may, if we wish, also define certain states of the system in terms of what has happened and what is to happen, using tense-logical operators Sometime-Past, Always-Past, Sometime-Future and Always-Future. We have to be somewhat careful, however, because use of the tense-logical operators does complicate the logic of the situation somewhat: tense logic is much less well-developed in terms of usable automated or semi-automated tools and techniques than predicate or propositional logic. If we do use tense logic, we might wish to define the following meaning postulates for further properties that might turn out to be of interest:

Sent(msg,NIC)
$\Longleftrightarrow$
NOT Sending(msg,NIC) AND Sometime-Past(Sending(msg,NIC))

Received(msg,NIC)
$\Longleftrightarrow$
Sometime-Past(Receiving(msg,NIC)) AND NOT On(msg,Bus)

These two predicates are not symmetric. It should be obvious that a message that has been *Received* by its NIC is no longer *On* the bus. However, a message can have been *Sent* and still be in transit, so it is *On* the bus. Or it might have been *Sent* and already *Received*, in which case it is no longer *On* the bus. And we shall see, later, that not expressing the status of the message on the medium after it has been *Sent* will enable us to express the loss of a message, which we have already identified as a hazardous happenstance.

We should keep in mind that the point of the exercise will be to identify and analyse hazards that occur with the communications bus, which is concerned with the danger of real-life use, and not to axiomatise all properties of the bus, which is a maybe useful exercise for developing a facility with logical expression, but rather more work than may strictly be needed to find out how things may go dangerously wrong.

3.4 OPRA, Refinement and OHA of Train Dispatching

3.4.1 Introduction

In his PhD thesis, Bernd Sieker formalised the German train-dispatching protocol for non-state-owned railways ("Zugleitbetrieb") using Ontological Hazard Analysis (OHA) [17]. Briefly, OHA involves

- OPRA (Level 0)

- performing a hazard analysis using this Level-0 OPRA
- formulating safety requirements at Level 0
- using formal refinement to generate a further OPRA (Level 1)
- performing a hazard analysis using Level-1 OPRA
- assuring the safety requirements formulated at Level 0 are assured at Level 1 (here, by formal proof using formal logic)
- formulating new Level-1 safety requirements should they be necessary
- iterating this process until done.

Our focus here is on the sequence of OPRA levels which are created in this analysis. However, it is worth indicating what other considerations come into play in this development, for example the logical analysis and logical proof which is used to generate the original safety requirements at Level 0, and to ensure that the safety requirements at a further Level logically imply the safety requirements formulated at Level 0 (traceability). Some of this will be indicated here. Also pertinent are the informatics technologies which come into play in the analysis, such as formal logic, formal refinement, state machines, message flow graphs - all technologies which might be familiar to informaticians but completely foreign to most railway engineers. Being successful in devising reliable digital systems for railways really does require understanding formal logic, formal refinement, state machines, message flow graphs, the radio cell-based communication protocol GSM-R based on the cell-phone network GSM protocols, encryption-key management, the Eurobalise electronic milestones, the European Train Control System specification (ETCS) and the European Rail Traffic Management System (ERTMS, in short ETCS+GSM-R). It's getting to be a tough world!

German administrative law [18] sets the requirements for how train dispatching is to be performed on non-state-owned railways. Sieker derived a system, expressed in SPARK source code, which provably implements a (completed version of) this legal protocol. This process is described here. It seemed useful to go through the derivation, because not only is Level 0 fairly trivial and Level 4 fairly complicated, but the development of the Levels and derivation of the safety requirements illustrates the formal refinement process in OHA by means of a relatively clean example.

Complete traceability is maintained in this example between the abstract high-level safety requirements and the SPARK source code through the formal refinement. The importance of this traceability is as follows. Were the SPARK code to be implemented in communicating digital-computing machines which back up (or even replace!) the human agents of the system, then this analysis of the system demonstrates that the logic of the communications, as expressed in these machines, is faultless. This contrasts favourably with occasional human (mis)performance of these protocols (consider the Warngau accident [13]).

There remains residual risk, of course, in the risks associated with the ADA compiler used on the SPARK source code, the transfer of this object code to the target hardware, faults associated with 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* (equivalently, assertions modulo logical equivalence) for two trains in the available language, and determining which of these are safety requirements. It is by no means the case that this can be done for each and every example. For one to be able to do so, it is likely necessary that the Level 0 OPRA, as in this case, is very, very simple.

The original dispatching protocol ("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 in progress. The system, as well as its derived system developed here, relies solely on messages passed between the train conductors and the dispatcher.

3.4.2 Ontological Hazard Analysis

We illustrate not only the OPRA, but the refinements involved in the OHA and the derivation of the safety requirements, as an introduction to OHA.

Structure of the Analysis

Figure 3.5 shows the structure of the refinements and traceability proofs of the Ontological Hazard Analysis performed for this case.

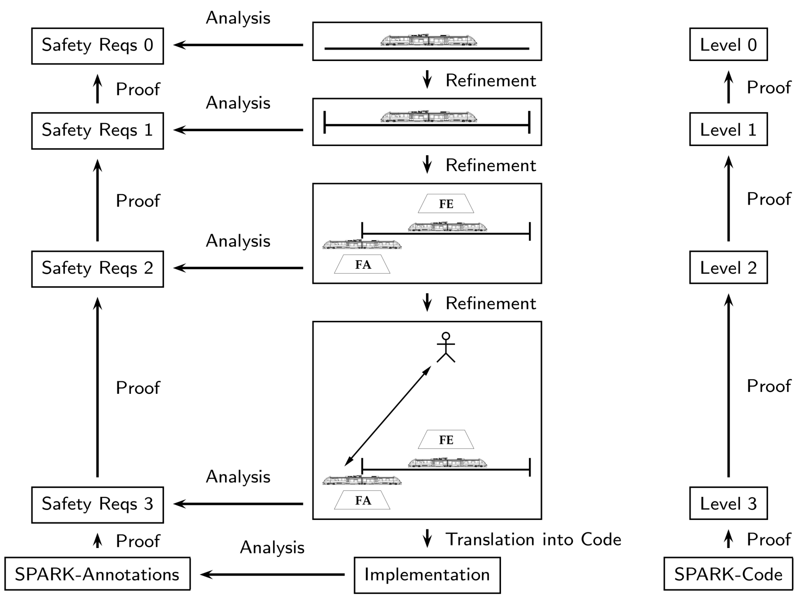


Figure 3.5: Structure of the OHA

OPRA Analysis at Level 0

The goal of the OPRA analysis at Level 0 is not to provide a detailed description of train operations, but rather to provide a description that is so simple that

- we can define safety axioms to which all applications experts can assent, and
- ascertain that these axioms are both *correct* and *complete* relative to the expressions of the language.

The train dispatching protocol, in fact most rail operational protocol, divides the track into sections, called blocks. Here, the blocks are fixed and permanent, as they are in most rail operations. The object of a dispatching protocol is to ensure that no two trains occupy the same block, partially or completely, at the same time, in normal operations, known as “operations under central responsibility” (i.e., that of the controller/dispatcher). Access to blocks is normally controlled by signals, but some lines are used infrequently enough that it is not cost-effective to install signals. Such lines are controlled by human dispatching, as we are considering here.

Trains can also move under “line of sight” operations, in which train drivers can

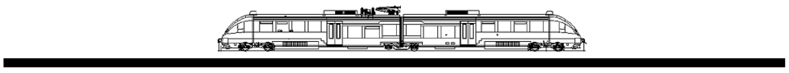


Figure 3.6: Schematic Representation of Level 0

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

Table 3.1: Level 0 Sorts

Relation	Description
inA(F,S)	Train F is in Block 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)

Table 3.2: Level 0 Relations

halt their trains within the space that they can see to be clear, taking into account the track layout. Line of sight operations take place usually at a very low speed, from walking pace up to 30-40 km per hour.

There are some low-speed urban rail systems that use moving blocks, which move along with a train. Such moving-block protocols ensure that no two moving blocks intersect.

So the purpose of Level 0 is to be able to express that no two trains can occupy the same block at the same time. The types of objects (here called *sorts* in the vocabulary of formal logic) are thus vehicles and blocks, as in Table 3.1. The relations needed for expressing this mutual exclusion property are listed in Table 3.2.

Determining Safety Axioms

We use the formal language of quantifier-free predicate logic to express the safety requirements. The basic vocabulary, the object sorts (properties) and the so-called "atomic propositions", is given in Tables 3.1 and 3.2. We need to say that no two trains can be in the same block at the same time. To do that, we introduce two constants for objects of sort "vehicle", namely *F1* and *F2*, and one constant *S* for one object of sort "block".

There are just a certain number of non-equivalent statements in propositional logic which one can formulate concerning *F1*, *F2* and *S* using the properties and relations in Tables 3.1 and 3.2. It is fairly straightforward to list initially 256 statements, to one of which any statement is obviously equivalent, and from those 256 statements a certain amount of elementary propositional-logical manipulation reduces these to a couple of dozen non-equivalent statements. Domain knowledge (the intended meaning of the symbols in the railway application) results in just 6 statements that can be regarded as relating to safety. This process is described in detail in Section 3.3 of [17] but omitted here, because this volume is not a treatise on elementary logic. The 6 statements related to safety, the safety axioms of the development, are listed in Table 3.3. We use the following shorthand notation for a train F1 and one block S: LV(F1,S) is written as LV1, ZV(F1,S) is written as LZ1, and inA(F1,S) is written as in1; similarly for train F2.

Level 1: First Refinement

The generic block of Level 0 is refined as follows, introducing the new sorts Track and Station. The purpose of this is that there are two kinds of blocks laid down by the law of German railway operations, a free-track block and a station. This leads to the sorts found in Table 3.4. The additional relations are in Table 3.5. 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.

The meaning postulates are as follows (where I use the usual terminology $\wedge$ for logical-AND, $\vee$ for logical-OR, and $\exists$ for the existential quantifier, "there exists"). The first postulate says that a Block (from Level 0) is either a Freeblock or a Track:

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.

Table 3.3: Safety Postulates at Level 0

$$Block(S) \Leftrightarrow$$

$$(\exists A \exists B(Station(A) \wedge Station(B) \wedge terminates(S,A,B)) \vee Track(S)$$

The second defines when a train is in a block, by distinguishing the cases of when it is in a Freeblock and when in a Track:

$$Block(S) \wedge Train(F) \wedge inA(F,S) \Leftrightarrow$$

$$((\exists A \exists B(Station(A) \wedge Station(B) \wedge terminates(S,A,B) \wedge between(F,A,B))\vee$$

$$(Track(S) \wedge inG(F,S)))$$

The third postulate defines when a train is under central control, namely (again) either when it is under central control in a Freeblock or when it is under central

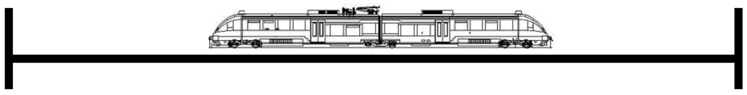


Figure 3.7: Schematic Representation of Level 1

Vehicle	Train or other track vehicle
Freeblock	A “free” track section outside of a station
Track	A piece of track in the station
Station	A station where messages are exchanged

Table 3.4: Sorts in Level 1

Relation	Description
inG(F,S)	Train F is in station Track S
ZG(F,G)	Train F may occupy station Track G under central responsibility (normal scheduled-operation)
inG(F,G)	Train F is in station Track G
inZ(F,A)	Train F is at station A
terminates(S, A, B)	Stations A and B terminate block S
between(F,A,B)	Train F is between stations A and B
ZZ(F,A,B)	Train F may travel between stations A and B under central responsibility

Table 3.5: Level 1 Additional Relations

control in a Track:

$$Block(S) \wedge Train(F) \wedge ZV(F,S) \Leftrightarrow$$

$$(\exists A \exists B(Station(A) \wedge Station(B) \wedge terminates(S,A,B) \wedge ZZ(F,A,B)) \vee (Track(S) \wedge ZG(F,S))$$

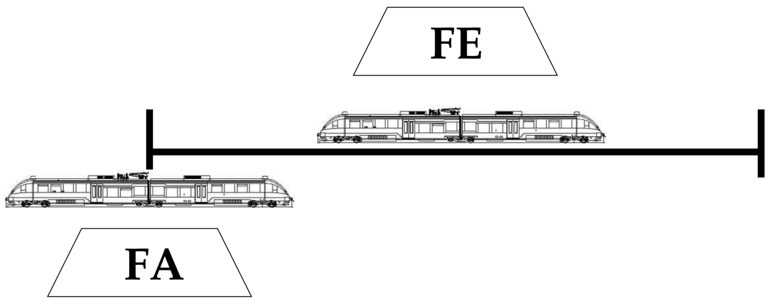


Figure 3.8: Schematic Representation of Level 2

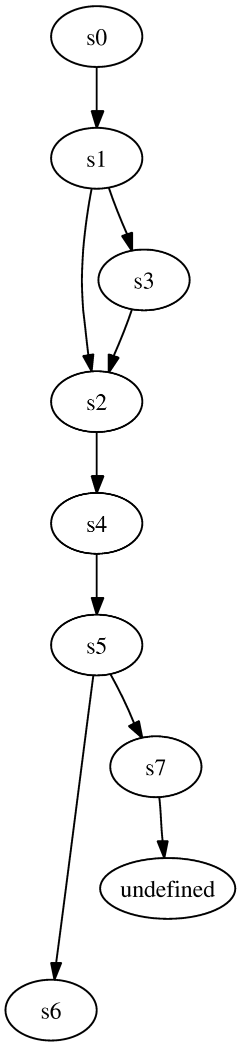
FA(F,A,B)	Train F, in station A, has asked for clearance to go to station B
FE(F,A,B)	Train F, in station A, has received clearance to go to station B
AFE(F,A,B)	Train F, in station A, has been denied clearance to go to station B
KH(F,A,B)	No obstructions are known for train F to go from station A to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

Table 3.6: Relations in Level 2

Level 2

In Level 2 no new sorts are added. Additional relations concerning requesting and receiving "clearances" (permissions to enter a block) are added. We are now able to build a state-machine representing the global states of clearances which represents a train journey. The state-machine looks as in Figure 3.9, which is presented as a Predicate-Action-Diagram [11]. There are in addition three simple Meaning Postulates, which I don't show

At this level, some elementary logic leads to two new Safety Postulates. These are requirements which need to be fulfilled at this level in order to ensure that the safety requirements at the previous levels, Level 0 and Level 1, can be proved. This condition is known as the *traceability of the Level 0 and Level 1 requirements at Level 2*. The two Safety Postulates can be expressed informally as:



(a) Predicate-Action-Diagram

s0	$\mathrm{inZ}(F,A)$
s1	$\wedge\ \mathrm{inZ}(F,A)$ $\wedge\ \mathrm{FA}(F,A,\mathit{Next}(F,A))$ $\wedge\ \neg\,\mathrm{FE}(F,A,\mathit{Next}(F,A))$
s2	$\wedge\ \mathrm{inZ}(F,A)$ $\wedge\ \mathrm{FA}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{KH}(F,A,\mathit{Next}(F,A))$
s3	$\wedge\ \mathrm{inZ}(F,A)$ $\wedge\ \mathrm{FA}(F,A,\mathit{Next}(F,A))$ $\wedge\ \neg\,\mathrm{FE}(F,A,\mathit{Next}(F,A))$ $\wedge\ \neg\,\mathrm{KH}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{AFE}(F,A,\mathit{Next}(F,A))$
s4	$\wedge\ \mathrm{inZ}(F,A)$ $\wedge\ \mathrm{FA}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{FE}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{KH}(F,A,\mathit{Next}(F,A)$
s5	$\wedge\ \mathrm{zw}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{FE}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{KH}(F,A,\mathit{Next}(F,A))$ $\wedge\ \neg\,\mathrm{LV}(F)$
s6	$\mathrm{inZ}(F,A)$ $=s0$
s7	$\wedge\ \mathrm{zw}(F,A,\mathit{Next}(F,A))$ $\wedge\ \mathrm{FE}(F,A,\mathit{Next}(F,A))$ $\wedge\ \neg\,\mathrm{KH}(F,A,\mathit{Next}(F,A))$ $\wedge\ \neg\,\mathrm{LV}(F)$

(b) States of the PDA

Figure 3.9: The State Machine for Level 2

- 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.

We will not consider the safety requirements or the traceability of the safety requirements through the refinement further in this chapter. The main point here is the OPRA and its connections across the levels of the refinement.

Hazards

There are potentially new hazards identified at every level of the refinement. At Level 2, the newly-identified hazards this 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.

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 [18].

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:

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

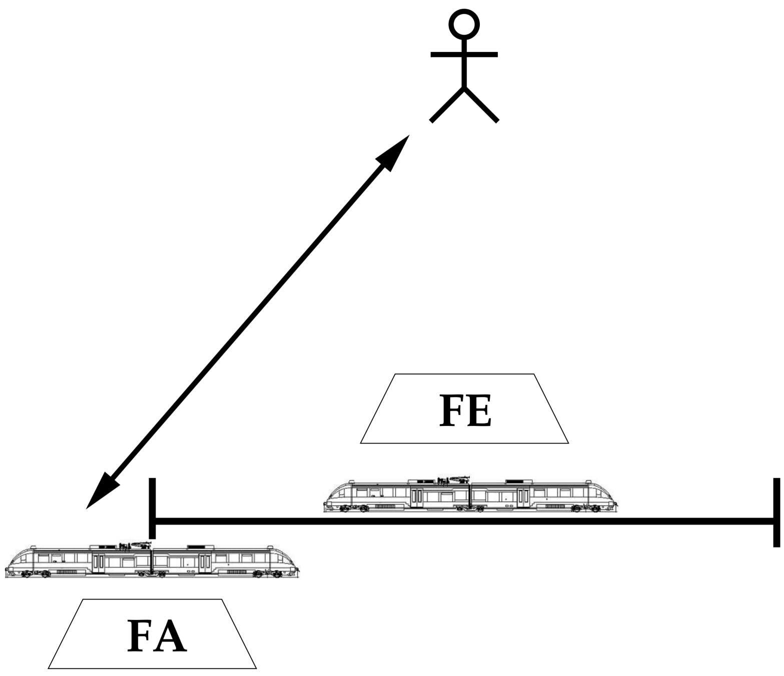


Figure 3.10: Schematic Representation of Level 3

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

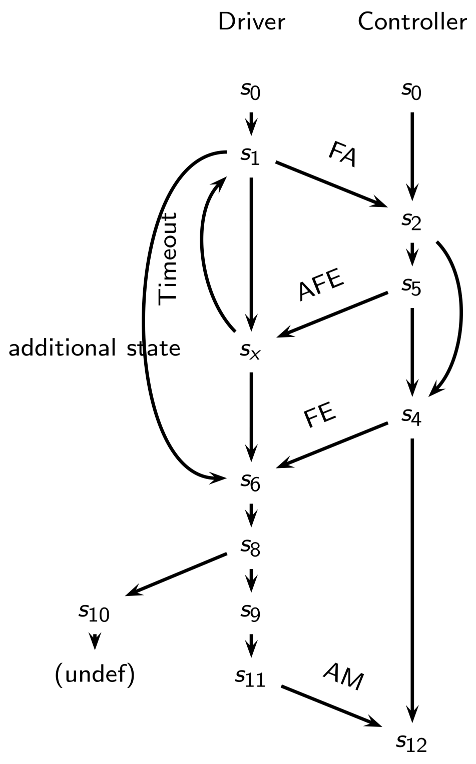


Figure 3.11: The Message Flow Graph

the SPARK verification conditions are determined by hand to define the MFG as here pictured.

MFG-Trans.	Driver-State	Controller State	Global State
s0	inZ(T,A)	–	inZ(T,A)
s0 $\rightarrow$ s1	inZ(T,A) $\wedge$ Sent$\langle$FA,T,Next(T,A)$\rangle$	--	inZ(T,A) $\wedge$ Sent$\langle$FA,T,Next(T,A)$\rangle$
s1 $\rightarrow$ s2	--	Recd$\langle$FA,T,Next(T,A)$\rangle$	inZ(T,A) $\wedge$ Sent$\langle$FA,T,Next(T,A)$\rangle$ $\wedge$ Recd$\langle$FA,T,Next(T,A)$\rangle$

Table 3.7: Example Definitions of States and Transitions of the PDA for Level 3

3.5 The Step to Code: A Comment on the Implementation in SPARK

SPARK is based on a subset of the Ada language. It uses annotations to denote data and information [1] 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. Code for train dispatching was written by Phil Thornley of SparkSure, based on the Message Flow Graphs. Proofs were completed that the Code fulfills the annotations, and that the annotations fulfill the Level 3 Message Flow Graph description. The uninterrupted requirements tracing (the "Safety Postulates") from Level 0 requirements down to the SPARK source code ensures that the source code fulfils the Safety Requirements of Level 0.

Bibliography

[1] John Barnes with Praxis, *High Integrity Software: The SPARK Approach to Safety and Security*, Addison-Wesley, 2003/2006.

[2] Robert Bosch GmbH, *Automotive Electrics and Automotive Electronics: Systems and Components, Networking and Hybrid Drive*, 5th Edition, Springer-Vieweg, 2007 (German), 2013 (English).

[3] Kevin Driscoll, *Murphy Was an Optimist*, ongoing lecture series. Version 19 of the lecture (circa 2010) is available from https://rvs-bi.de/publications/DriscollMurphyv19.pdf, no date.

[4] Kevin Driscoll, Brendan Hall, Håkan Sivenkrona and Phil Zumsteg, *Byzantine Fault Tolerance, from Theory to Reality*, in Computer Safety, Reliability and Security, Proceedings of the 22nd International Conference, SAFECOMP 2003, Lecture Notes in Computer Science volume 2788, Springer-Verlag, 2003. Available from https://www.cs.indiana.edu/classes/p545-sjoh/post/lec/fault-tolerance/Driscoll-Hall-Sivencrona-Xumsteg-03.pdf , accessed 2017-06-14.

[5] R.W. Hazell, G.V. McHattie and I. Wrightson, *Note on Hazard and Operability Studies [HAZOP]*, Royal Society of Chemistry, London, 2001.

[6] International Electrotechnical Commission, IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems, 2nd Edition, 7 parts, 2010.

[7] Peter Bernard Ladkin, *Causal Reasoning about Aircraft Accidents*. In: Koornneef F, van der Meulen M (eds) Computer Safety, Reliability and Security, Proceedings of SAFECOMP 2000, Lecture Notes in Computer Science 1943:344-360, Springer-Verlag, 2000. Available in condensed form at https://rvs-bi.de/publications/books/CausalSystemAnalysis/Chapter_11_Accident_analysis_why_because_analysis.pdf

, accessed 2017-06-14.

[8] Peter Bernard Ladkin, *Ontological Analysis*, Safety Systems 14(3), Safety-Critical Systems Club, 2005.

[9] Peter Bernard Ladkin, *Causal System Analysis*, electronic edition, RVS 2001. Available from https://rvs-bi.de/publications/books/CausalSystemAnalysis/index.html , accessed 2017-06-14.

[10] Peter Bernard and Stefan Leue, *Interpreting Message Flow Graphs*, Formal Aspects of Computing 7(5):473–509, 1995. Available from https://rvs-bi.de/publications/abstracts.html#FAC-MSC .

[11] Leslie Lamport, *TLA in Pictures*, IEEE Transactions on Software Engineering SE-21:768-775, 1995. Available at http://lamport.azurewebsites.net/pubs/pubs.html#lamport-pictures , accessed 2017-06-14.

[12] J. L. Mackie, *The Cement of the Universe: A Study of Causation*, Oxford University Press, 1974.

[13] RVS Group, *Train Crash Near Warngau, Germany*, course material, RVS Group, University of Bielefeld, 2008. Available from http://wwwhomes.uni-bielefeld.de/cgoeker/SysSafe/WiSe%2011-12/Cases/TrainHeadOnCollisionWarngau.pdf

[14] Andrew S. Tanenbaum and David J. Weatherall, Computer Networks, 5th edition, Prentice-Hall, 2011.

[15] W.E. Vesely, F.F. Goldberg, N.H. Roberts and D.F. Haasl, *Fault Tree Handbook*, NUREG 0492, U.S. Nuclear Regulatory Commission, 1981. Available from https://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/

[16] Felix Redmill, Morris Chudleigh and James Catmur J, *System Safety: HAZOP and Software HAZOP*, John Wiley & Sons, 1999.

[17] Bernd Sieker, Systemanforderungsanalyse von Bahnbetriebsverfahren mit Hilfe der Ontological Hazard Analysis am Beispiel des Zugleitbetriebs nach FV-NE, Doctoral Dissertation (in German), RVS Group Tech-Fak and CITEC, Uni Bielefeld, April 2010. Available from https://rvs-bi.de/publications/Theses/Dissertation_Bernd_Sieker.pdf , accessed 2017-06-14.

[18] VDV (2004), *Fahrdienstvorschrift für Nicht-bundeseigene Eisenbahnen (FV-NE)*, (english: *Operating Regulations for Non-Federal Railways*), Verband Deutscher Verkehrsunternehmen (VDV) Ausgabe (edition) 1984.