CHAPTER 6

Case Study: Causal Fault Analysis of an Automobile Communications Bus: Level 0

6.1 Introduction

There are many introductory hazard analyses (HazAn) of engineered system designs to be found in safety textbooks. A pressure vessel with relief system is considered in [4]. A pressure vessel is also the first example in [5] which then includes a small motor overheating example and follows with a running example of a pressurised-water reactor, interspersed with a small example of a departure-monitoring device for a single-track railway. An event tree from an analysis of the Cassini spacecraft mission is shown as an example in [1], but the first worked-through examples are of an example reactor protection system, and the Storm Surge Barrier in the Rotterdam waterway system. Their third example is of a partially-redundant electrical supply system. Human operation and human error is considered in all three.

While these examples illustrate techniques developed over decades for hazard and risk analysis, it is typical that examples from the process industries are used, for this is often where such techniques are either pioneered or matured (Fault Tree Analysis started with the Minuteman ICBM systems in the US, but is more well-known from its use in the nuclear-power industry, for example [14]). The general techniques are not subject-matter-specific, but the characteristics of mechanical or electrical systems admit the introduction of specific techniques, such as statistical analyses, which work with the general properties of those systems, for example

reliability analyses of mechanical system components, whose failure modes are mostly known and surveyable. In contrast, statistical analyses of designs such as programmable electronic components whose behavior is driven by often-complex software is notoriously difficult. Hard constraints bound the assessment of the reliability of software through testing to a probability of failure per operational hour of $O(10^{-5})$ [10], whereas the desired reliability of the component of which the programmable electronics is part may well be in the range of $O(10^{-9})$ or lower.

There are some books which describe the adaptation of preferred methods in other industries to programmable systems, for example [12] describes the adaptation of the process-industry method of Hazard and Operations Analysis, HAZOP, to the assessment of software.

6.2 Ontological Hazard Analysis and Causal Fault Analysis

One of the issues with performing hazard analysis for software-based digital components of systems is that rarely do these components exhibit dangerous behaviour in and of themselves. They are connected to devices with safety-critical behaviour, and are often intended to control those devices, and it is the behaviour of the devices which is dangerous or not. The software-based digital component may induce this dangerous behaviour of the device, or not, and thus the behaviour of the digital system which induces this dangerous device behaviour can inherit the appellation as "dangerous".

However, when the software-based digital system is considered independently of devices which it controls, there are rarely, mostly no, functional behaviours which can appropriately be labelled "dangerous". Considered by itself, we are mostly concerned with whether the software-based digital system operates according to its requirements, its expectations. When it does not, we say there has been a *failure*. In common engineering terminology, a *fault* is a device state which causes a failure.

Here, we consider a generic communications bus for road transport vehicles. The usual techniques of potential hazard elicitation may be applied (we use HAZOP here [12]). Since we consider the bus stand-alone, there is no dangerous behaviour specified for the communications bus per se. Our analysis is thereby not properly a *hazard* analysis, since without dangerous behaviour there can be no hazards. However, we are searching for anomalous behaviour, failure behaviour which indicates a fault.

This can be accomplished using exactly the same techniques as in OHA, but in this case we are performing *failure analysis* rather than hazard analysis. We call the OHA techniques applied to failure analysis Causal Fault Analysis (CFA). We are cognisant, though, that a failure of function might well be a hazard in some application in which that function is critical, so we continue to use the term by calling a functional failure which we identify a *hazardous happenstance* or HazHapp.

The CFA is performed here at the high level to some level of detail. This is a generic analysis for a generic communications bus. It can thus serve as a reference model for such failure analyses of communications devices. Indeed, the analysis is not even bound to the bus as being a physical object. If radio communications are considered, it could even be the "ether".

6.2.1 Proceeding with CFA

CFA works on a hierarchy of abstractions, in the same way in which hierarchical decomposition is used in software design [11], and in digital system design in general, in order to obtain similar benefits. Just as waveforms are interpreted as bits, and sequences of bits are interpreted as bytes, and bytes are interpreted as various complex data structures; or bytes are interpreted as assembly-language commands, and sequences of assembly language commands are used to implement memory-variable assignments, loops, if-then-else statements, or Horn-clause declarative programs in Prolog, and so on, and these programs in turn are regarded as implementing a higher-level specification of what the software should do, so CFA works with a similar hierarchy, guided not by ease of general expression (as in SW-design), but by ease of expression specifically of HazHapps. Thereby we control the complexity of the PHA in the same manner in which it is controlled by hierarchical decomposition in SW specification and design. We assume some level of familiarity with hierarchical design and decomposition [11].

CFA proceeds by refinement, just the same as in OHA. HazHapps obtain names in the vocabulary. Some of the HazHapps may be expressible in terms of the vocabulary to hand, in which case meaning postulates containing these definitions are formulated. Some of them may not be so expressible. Further refinement steps will be necessary to express them. During the process, some convenient assumptions ultimately constraining the design of the artifact may be made, such as for example that a data packet contains an integral number of fields, and fields are always of the

form <attribute,value>. These constraints must be met in any bus design for which the analysis is to be valid.

At the end of a refinement stage we have

(a) an explicit vocabulary, the refinement-stage *primitives*

(b) a list of hazardous happenstances (events or states), *HazHapps*, with associated *Reliability Requirements* or *RelReqs*, namely mitigation and avoidance techniques), if such have been identified already,

(c) a list of *Meaning Postulates*, definitions of vocabulary which have been introduced during the course of the CFA,

(d) *assumptions* made during the course of the analysis, typically about the form of data objects, which simplify the analysis,

(e) *introduced vocabulary* (new primitive vocabulary), as yet undefined, needed for expressing the HazHapps identified at this stage.

The goal during a stage, which I call a *Level* as before, is to be as complete as possible in compiling (b), in particular in identifying RelReqs; to get as many of the new primitives defined in terms of the Level primitives by meaning postulates as possible; and to keep the number of new primitives to a minimum.

When the Level is finished, a new refinement Level must be chosen, and then worked through similarly. A new refinement Level is chosen by selecting a HazHapp, or a collection of intuitively related HazHapps, from the list in (b) which we can call the focus HazHapps for the new refinement Level.

The focus HazHapps are expressed as far as possible by means of the new primitives in (e) and through additional new primitives introduced for the purpose (and added to those already in the list in (e)). The goal will be, as in the previous Level, to identify RelReqs associated with the focus HazHapps, or to reformulate intuitively-similar focus HazHapps in terms of new primitives and meaning postulates which capture their common features.

A subsidiary goal at any Level is simultaneously to reduce the new primitives as far as possible through deriving meaning postulates. When the CFA reaches a Level at which most of the identified HazHapps have associated RelReqs, it can be terminated. Those HazHapps without associated RelReqs must be retained and analysed at a later HazAn Level, or which there will typically be many in a complete system development. The assumptions listed in (d) must be adhered to, or eliminated, through further

development, at pain of invalidating the analysis.

6.2.2 A Generic Digital-Communication Bus

The generic digital-communication bus is illustrated again in Figure 1. In an automotive application, whence this example was abstracted, sensors measure current values of such phenomena as force at wheels, rotational speed of wheels, position of steering frames, and braking. The sensors produce digital values which are transmitted via the bus to command elements 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 information from a single device which is transmitted as a unit will be called a *message*. Each device, sensor, controller or actuator, is connected to the bus through a management device called a *Network Interface Controller* (NIC). Among the tasks of a NIC is to format the information in a form appropriate for bus transmission, 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. It also synchronises 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, when two NICs try to transmit almost simultaneously.

At this point, according to the "life cycle" of IEC 61508 (see [7, Flow Chart, p9]), one may 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.

It is sometimes important to distinguish 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]. Here, it will first be helpful not to distinguish: let me call either a state or an event a happenstance. Some happenstances can be hazardous in some systems. For example, suppose a command message to an actuator to steer left is corrupted or lost in transit. Likely this will result in a failure of the vehicle to

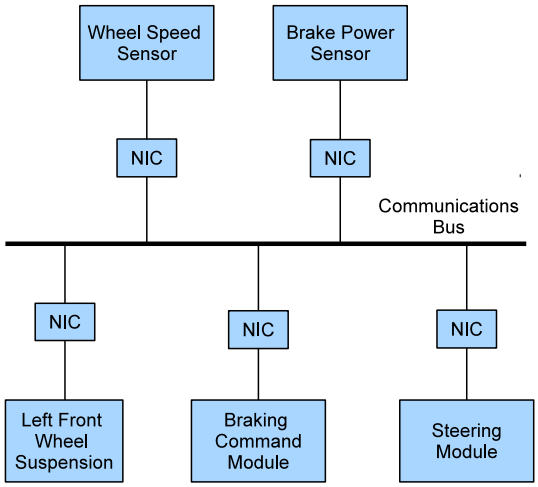

Figure 6.1: A Communications Bus

steer left. A failure to steer left on command 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. So some happenstances can be hazardous, and without a specific application is is not possible to say which. I shall thus refer to all of the failure happenstances as hazardous happenstances.

This terminology is particularly justified if the bus is the single point of communication of control actions. 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, we can foresee that the brakes will not be applied. There is a prima facie intuitive reason for calling it 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 loss of a message is a HazHapp.

We may imagine that if a command to brake harder arrives at the brake actuator

and is read as to 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, is a HazHapp.

Not only may loss of a message and corruption of a message be a HazHapps, so might the incorrect reading of a message by a NIC. These may not be the only HazHapps, but they are a start for the CFA.

We commence OHA or CFA by asking 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. It follows that we are speaking of the informational content, the digital interpretation, of whatever passes along the medium as the message is transmitted.

It is well to distinguish here information from data. The information is, if you like, the meaning of the message. The data is the particular form which the information takes - maybe considered as a precise physically continuous waveform which is actually put on the medium by electronic means by the NICs. Or the data could be considered as a sequence of 0's and 1's into which the waveform is interpreted by the NICs at each end of the transmission. Both of these representations of data are obviously distinct from the meaning of the message, the information it conveys.

Further, we might know there can be arbitration difficulties, such as when a waveform lies on the boundary of what counts as a "1" bit and what counts as a "0". 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 [1]. So we can decide at this point if we are dealing with messages as physical waveforms, or whether we are dealing with messages as sequential, structured information, containing values in data types, protocol information, and so forth, which may be considered, in the usual digital reduction, to be sequences of bits.

6.3 A Little Elementary Logic and Notation

In this and the following chapter, some syntactic facility with elementary logic and its notation will help, for example the following.

- I write $\forall x.A$ with a "." to mean $\forall x(A)$.
- I write $\forall x : X.A$, where X is an object type (known in logic rather as a *sort*, a unary predicate; the word "type" means something else), to mean $\forall x(X(x) \Rightarrow A)$
- $\forall x : X \forall y : Y.A$ thus means $\forall x(X(x) \Rightarrow \forall y.(Y(y) \Rightarrow A))$ which is in turn equivalent to $\forall x \forall y(X(x)$ AND $Y(y) \Rightarrow A)$.
- An n-ary function can be represented by an (n+1)-ary relation with a meaning postulate (a constraint), namely $F(x) = y$ can be written as $F'(x,y)$ along with the postulate ($F'(x,y)$ AND $F'(x,z) \Rightarrow y = z$). Although in the acronym OPRA there is no term for "Function", I shall use function notation willy-nilly with the understanding as here that it is equivalent to predicates with a functional constraint if this is desired instead.
- I write a predicate $A(x,y)$ where the first argument x has type X and the second argument y has type Y quite often as $A(X,Y)$ rather than the more usual (in computer-scientific writing) $A(x{:}X, y{:}Y)$. Both mean $A(x,y)$ AND $X(x)$ AND $Y(y)$.
- I write "A WHERE B" sometimes, which means in logical syntax "B $\Rightarrow$ A".

6.4 Level 0

We call the highest level 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 hierarchical design, a natural number (non-negative integer in mathematics) may be defined in terms of a non-negative-integer number range. Then an integer number range may be defined in terms of sequences of bits, with operations on bits implementing the mathematical operations on integers. To show that the implementation is correct, it must be proved – verified – that the operations on bits indeed implement the operations on integers at the higher (mathematical) level. Similarly, it must be verified in CFA 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 as bits, then its description is much simpler. We must say only what bit sequences are to be carried, and not be concerned at this Level with how this is carried out.

We have considered that 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, 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 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 by the information-originator (only the first was sent by this originator). Whether messages could be generated in the system, say, by outside electrical influences on the medium, is a matter for more detailed consideration at later Levels.

We can now define the formal OPR vocabulary at Level 0.

6.4.1 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 rather than names for specific objects, although *Bus* could be considered as one object, since there only one of them.

6.4.2 Properties

I consider the properties of each object type in turn.

Bus The bus itself transmits messages. This is a relation between the bus and 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)*, but I do not do so.

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 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.

- In particular, if the NIC becomes unattached from the Bus, then it has lost integrity. I introduce a relation Attached(NIC,Bus).
- What if the NIC becomes unattached from its associated device? The associated device is not an object in our list of objects. We can indicate this by the introduction of a new property of a NIC: AttachedToDevice(NIC) if it is satisfactorily attached to its associated device.

Correspondingly, we have the following two properties: *Integrity(NIC)*, *Attached(NIC)*. Again, we could write *Integrity(Y: NIC)* and *Attached(Y: NIC)*, but it seems to me that we can use obvious names such as NIC1, NIC2, etc, to indicate both an object and its type at this level of discourse, without being too much concerned about exact syntax.

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

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

Objects	Properties
Bus	Integrity
NIC	Integrity AttachedToDevice
msg	Content Size Deadline

Figure 6.2: Level 0 Object Types and Their Properties

6.4.3 HazHapps So Far

We have already identified some HazHapps in this initial discussion. They are shown in Figure 6.3.

NOT-Integrity(Bus)
NOT-Integrity(NIC)
NOT-AttachedToDevice(NIC)

Figure 6.3: Some Initial HazHapps from Properties at Level 0

6.4.4 Relations

A NIC may be attached or not to the bus. I 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 bus medium. A message may be *in* a NIC when it is being assembled or disassembled.

We 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 6.4, in which an X indicates that the relation has the object type as an argument. An additional HazHapp has resulted from our immediate consideration of the relations, in Figure 6.5.

Figures 6.2 and 6.4 constitute the definition of the vocabulary for (a) of Level 0. This vocabulary will be extended forthwith.

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

Figure 6.4: Level 0 Relations Amongst Object Types

NOT-Attached(NIC,Bus)

Figure 6.5: An Additional HazHapp from the Relations at Level 0

6.4.5 Meaning Postulates

So far, the vocabulary is just words. But words have meanings, in particular the words used have particular intended meanings for the bus and associated NICs and msgs. The property of being *Sent* or being *Received* by a *NIC* entails that a *msg* is at the same time *On* the *Bus*. It also entails that it is not yet *In* the receiving *NIC*. This is part of what we mean by asserting *Sent* or *Received*[1]. It could be otherwise: I 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 I did not choose this second option; I chose the first. I have to mark this distinction somehow – we have to say what we mean by use of the words. We do this in part in OHA and CFA through enumerating logical relations between the relations, properties and objects. These logical assertions are called *meaning postulates*.

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

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

1 I shall not use italic script further for formal OPR elements. I shall capitalise property and relation names.

We have also noted that when a NIC becomes unattached to its associated device or to the Bus, the Integrity of the NIC has been lost. This can be expressed using the partial meaning postulates:

$$\text{NOT AttachedToDevice(NIC)} \Rightarrow \text{NOT Integrity(NIC)}$$
$$\text{NOT Attached(NIC,Bus)} \Rightarrow \text{NOT Integrity(NIC)}$$

We can modify these logically. In classical propositional logic, the rule of contraposition is valid, which says that

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$$

We could thus reinterpret the two partial meaning postulates more simply:

$$\text{Integrity(NIC)} \Rightarrow \text{AttachedToDevice(NIC)}$$
$$\text{Integrity(NIC)} \Rightarrow \text{Attached(NIC,Bus)}$$

However, the visible connection with HazHapps thereby disappears.

We might be tempted to define certain states of the system in terms of time, what has happened and what is to happen, using tense-logical operators *Sometime-In-The-Past*, *Always-In-The-Past*, *Sometime-In-The-Future* and *Always-In-The-Future*, which has proven to be a great help in terms of expressing desirable properties of behavioural systems [9]. Two of these operators have symbols associated with them, namely

- *Sometime-In-The-Future*: $\Diamond$
- *Always-In-The-Future*: $\Box$

and we can use the following derived symbols:

- *Sometime-In-The-Past*: $\Diamond_P$
- *Always-In-The-Past*: $\Box_P$

Use of the tense-logical operators does complicate the formal logic somewhat: this form of tense logic is much less well-developed in terms of usable automated or semi-automated tools and techniques than predicate or propositional logic. However, tense logic has expressive advantages for requirements, which is what we are concerned with here. For example, we can define the following meaning postulates for further propertiest:

$$\text{Sent(msg,NIC)} \Leftrightarrow \text{NOT Sending(msg,NIC) AND } \Diamond_P\text{(Sending(msg,NIC))}$$
$$\text{Received(msg,NIC)} \Leftrightarrow \Diamond_P\text{(Receiving(msg,NIC)) AND NOT On(msg,Bus)}$$

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. 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 HazHapp, and will do so again when we apply the HAZOP guide words, the keywords, to the vocabulary developed so far.

We recall, though, that the point of the current exercise is to identify and analyse HazHapps that occur with the communications bus and not to axiomatise all properties of the bus, which is maybe a useful exercise for developing a facility with logical expression, but rather more than is strictly needed to find out how things may go wrong.

To summarise the OPRA so far, we have

- the definitions of the Objects, Properties and Relations
- new Assertions: the partial meaning postulates for
  - Sending
  - Receiving
  - NOT AttachedToDevice
  - NOT Attached
- new Assertions: the meaning postulates for
  - Sent
  - Received
- four HazHapps already

6.4.6 Using HAZOP

We have a vocabulary for expressing some high-level properties and relations of the bus and associated objects. We can now look to identify HazHapps which are describable with this vocabulary or with an extension of the vocabulary by applying HAZOP.

HAZOP is a technique which, in its original form, depends crucially on group-think

[12]. My research group has found that in CFA one person working alone, and then checking results with colleagues, can mostly bring those colleagues to consensus on hisher application of HAZOP. We surmise that this is because much of the work is performed by the refinement process, HAZOP being used primarily to provoke thought. If a hazard is missed at one Level, we have experienced that it is likely to turn up at a later Level. We have no theoretical grounds for this observation. The phenomenon does reduce the personpower resources needed to perform a satisfactory hazard analysis or failure analysis.

There are examples of OHA in which HAZOP turned out to be entirely superfluous to the HazAn, as we saw in Chapter 3 with the railway-control example, in which a demonstrably-complete set of RelReqs (there, SafeReqs) were derived at Level 0, and the formal refinement consisted of making the system more concrete while preserving the SafeReqs through the refinement.

Interpreting HAZOP Guide Words for the Level 0 vocabulary: Properties

We recall that HAZOP uses the “guide words" in Figure 6.6 as hints towards system properties which might be hazardous or lead to hazardous happenstance.

The first step in a HAZOP is to combine these guide words with system properties and relations to derive more properties and relations which may be associated with possible HazHapp. We can start combining them with the properties.

No applies to *Integrity(Bus)* and *Integrity(NIC)*. Either these devices retain their integrity, or they have lost it. Exactly what this might mean can be left to more detailed steps further down in the hierarchy. No other guide words seem to lead to obvious properties involving Integrity.

That is pretty much it for Integrity of both Bus and NIC. Moving on to Content(msg), Size(msg) and Deadline(msg), there is somewhat more to be said about the interpretation of the list of guide words. I consider first the combination with Content.

No-Content(msg) seems to be an assertion that the msg has no content. Does it mean that the msg has been lost? Maybe. It may still be that some formal details of the msg, say, its ID, are present, but that the substantial information which is important to the receiver is not longer present. We thereby obtain two new properties:

Guide Word	Interpretation
No	None yet
More	None yet
Less	None yet
As well as	None yet
Part of	None yet
Reverse	None yet
Other than	None yet
Early	None yet
Late	None yet
Before	None yet
After	None yet
Faster	None yet
Slower	None yet
Where else	None yet

Figure 6.6: HAZOP Guide Words

- Lost(msg) means the msg has disappeared
- Corrupted(msg) means that some of the msg content has been altered

More-Content(msg) seems to be an assertion that the msg contains additional material from that which was inserted by its compiling NIC. Has the msg been corrupted? That depends on whether the additional material is an integral copy of material that is already contained in the msg. Or may it be that additional, meaningless information has been added?

Less-Content(msg) seems to be an assertion that the msg has lost some of its original content. If that has happened, the msg has been corrupted, without doubt. To say this, we need a new property of a msg, namely Corrupted(msg)

As-well-as-Content(msg) doesn't seem to carry much of a meaning further than More-Content.

Part-of-Content(msg) seems to mean that the msg has lost some of its original content, i.e., the same as Less-Content(msg). I shall use Part-of-Content to express this.

Reverse-Content(msg) seems to mean that the contents of msg have been reversed. How could this be interpreted? If the message is based on attribute-value pairs (a pair <attribute, value of attribute>, such as <receive-status,ready> or <receive-status,occupied>, then it doesn't matter what order the attributes and values are sequenced in the message. On the other hand, if the message is based on fixed fields, where a value is associated with an attribute because of its sequence position in the message, then it could be that values are read as values to the wrong attributes. This would be a form of corruption of the msg. It could also be avoided by using resilient message formatting, such as attribute-value pairs!

Other-than-Content(msg) seems to mean that the content is other than it should be, i.e., that the msg has been corrupted. It could also mean that an extra message appears (On the Bus, or Received by a NIC) that has not explicitly been generated – a phantom, Phantom(msg).

The keywords **Early**, **Late**, **Before**, **After** seem more to relate to timing properties of the msg itself, and not its Content. It could be, of course, that specific fields in the msg do contain timing information, and this timing information could be distorted as the keywords imply. This would be a form of message corruption. To interpret this specific form of corruption would be a task for later Levels, when the msg has been broken down into its component parts in the hierarchical decomposition.

Where-else-Content(msg) might seem to mean that the msg has been interpreted (decomposed) by a NIC for which it was not intended. We could express this assertion by saying, in logic, that the msg has been received by some NIC that was not its intended receiver:

$$\text{InappropriateReceiver(msg,NIC1)} \Leftrightarrow \text{Received(msg,NIC1) AND NOT (IntendedReceiver(msg,NIC1))}$$

To be able to say this, we would need to introduce the new predicate IntendedReceiver.

In summary, combining HAZOP keywords with Content has led to the introduction of:

- the new undefined property Lost(msg)
- the new undefined property Corrupted(msg)

- the new undefined property Phantom(msg)
- the new undefined relation IntendedReceiver(msg,NIC)
- the new defined relation/meaning postulate InappropriateReceiver

It is intuitively clear that the new properties Lost, Corrupted and Phantom represent HazHapps of messages, as does inappropriate reception, so we have the additional HazHapps in Figure 6.7.

Lost(msg)
Corrupted(msg)
Phantom(msg)
InappropriateReceiver(msg,NIC)

Figure 6.7: HazHapps from HAZOP applied to Content(msg)

Relations or Functions?

Rather than consider IntendedReceiver as a relation, we might well consider it as a function, say

$$\text{IntendedReceiverF(msg)} = \text{NIC1} \Leftrightarrow \text{IntendedReceiver(msg,NIC)}$$

However, this might unduly restrict the form of expression: a msg might well have multiple intended receivers, and then a formulation as a function would not work. We could consider whether we need a function name for the Sender of a message. A message is created on the Bus when Sending, and it is the unique NIC which is Sending which is the sender. So the identity of the sender is fixed by the act of Sending, namely NIC1 is the sender of msg1 precisely when:

$$\text{Sender(msg1)} = \text{NIC1} \Leftrightarrow \text{Sent(msg1,NIC1)}$$

Continuing the HAZOP: Size(msg)

We have considered the combination of HAZOP keywords so far with Integrity(Bus), Integrity(NIC) and Content(msg). We complete the HAZOP by considering the combination of keywords with Size(msg) and Deadline(msg).

No-Size(msg) could mean that the message has no size, in other words all substantial content has been lost. However we take this, it seems to be the same as the interpretation of No-Content(msg), that is, either Lost(msg) or Corrupted(msg). Similarly, More-Size with More-Content, Less-Size with Less-Content, Part-of-Size with Part-of-Content. Reverse-Size seems to have little meaning. Other-than-Size seems to mean something similar to More-Size or Less-Size. The keywords Early, Late, Before, After, Faster, Slower and Where-else seem to have no obvious interpretation with respect to the property Size.

Thus we have not observed any phenomena through considering the combination of keywords with Size(msg) which we had not already noted.

Continuing the HAZOP: Deadline(msg)

Now to consider the combination of keywords with Deadline(msg). The ordering keywords would have an interpretation here: Early-Deadline(msg), Late-Deadline(msg), Before-Deadline(msg), After-Deadline(msg), Faster-Deadline(msg), Slower-Deadline(msg). The combinations here seem to suggest the following. Messages can have an early deadline as well as a late deadline: that is, they are to be processed within an Interval(msg) = <Earliest(msg),Latest(msg)>, and that the message can be received outside or partly outside this interval and thus not be processed within the intended time – recall our mention earlier of Byzantine failures [1]. This may perhaps be best expressed through two predicates OutsideInterval(msg) and PartlyOutsideInterval(msg), leaving the details of early/lateness and partially early/late for further steps in the hierarchical decomposition. These new predicates OutsideInterval(msg) and PartlyOutsideInterval(msg) are taken here to be primitive with the expectation that they will be defined in later Levels.

Considering the combination of HAZOP guide words with Deadline(msg), then, we have identified

- a property (or function) of a message, Interval(msg)
- an assertion OutsideInterval(msg)
- an assertion PartlyOutsideInterval(msg)

We further observed that OutsideInterval and PartlyOutsideInterval could be involved in hazardous happenstance, as shown in Figure **??** We have now completed the combination of HAZOP guide words with the Properties. It remains to consider the

Relations.

OutsideInterval(msg)
PartlyOutsideInterval(msg)

Figure 6.8: HazHapps from HAZOP applied to Deadline(msg)

Continuing the HAZOP: Relations

Considering the relations Attached, On, In, Sending, Receiving, I state without argumentation that the interpretable keywords are restricted to Not and Partly. The argumentation is reserved as exercises for the reader in Section 6.5. We have already considered Not-Attached. Partly-Attached may mean that the NIC appears only intermittently to be attached, to the medium. Partly-Sending, Partly-Receiving suggest that a msg is being incompletely put on the Bus, respectively read from the Bus, by the NIC. These appear to be the only happenstances which we may form using the keywords which do not occur normally in the course of normal operations. For example, Partly-On and Partly-In could well refer to a msg in course of being placed on or received from the Bus by a NIC

6.4.7 HazHapps: Summary and Discussion

The results of applying the HAZOP guide words applied to the Level 0 properties and relations are summarised in Figure 6.9. We have obtained some insight into what can go hazardously wrong. We have identified the following hazardous happenstances so far in Figure 6.10 and now consider them further.

NOT Integrity(Bus) The integrity of the Bus is a physical primitive. This HazHapp can be expressed using the usual logical operators, here the operator NOT, on this primitive term.

NOT Integrity(NIC) The integrity of the NIC is a physical primitive. This HazHapp can be expressed using the usual logical operators, here the operator NOT, on this primitive term.

NOT AttachedToDevice(NIC) The attachment to its associated device of the NIC is

Property	Guide Word	Object	Interpretation
Integrity	No	Bus	has lost integrity
	No	NIC	has lost integrity
Content	No	msg	has been lost, or lost complete content: Lost(msg)
	More	msg	has gained informational content: Corrupted(msg)
	Less	msg	has lost informational content: Corrupted(msg)
	Part-of	msg	same as Less-Content(msg): Corrupted(msg)
	Reverse	msg	?non-resilient msg formatting? otherwise Corrupted(msg)
	Other	msg	has been corrupted: Corrupted(msg)
	Where-else	msg	IntendedReceiver not reached
Size	No	msg	same as No-Content(msg): Corrupted(msg)
Deadline		msg	to be defined through Interval(msg), OutsideInterval(msg), PartlyOutsideInterval(msg)
Attached/	No, Partly		.....
On/In/Sending	No, Partly		.....
Receiving	No, Partly		.....

Figure 6.9: Interpretations of the HAZOP Guide Words

a physical primitive. This HazHapp can be expressed using the usual logical operators, here the operator NOT, on this primitive term.

NOT Attached(NIC,Bus) The attachment of the NIC to the Bus is a physical primitive. This HazHapp can be expressed using the usual logical operators, here the operator NOT, on this primitive term.

Lost(msg) We try to express this hazardous happenstance in the vocabulary we have. We start by considering what being lost might mean. The answer seems obvious:

NOT-Integrity(Bus)
NOT-Integrity(NIC)
NOT-AttachedToDevice(NIC)
NOT-Attached(NIC,Bus)
Lost(msg)
Corrupted(msg)
Phantom(msg)
InappropriateReceiver(msg,NIC)
OutsideInterval(msg)
PartlyOutsideInterval(msg)
?Partly-Attached(NIC,Bus)?
?Partly-Sending(msg,NIC)?
?Partly-Receiving(msg,NIC)?

Figure 6.10: HazHapps at Level 0

the message was sent but never received by the receiver. We have to distinguish this happenstance from the situation in which the message has been sent but not yet received, because it is still in transit on the Bus. We may express the difference using the vocabulary we already have, namely that in the second, normal, case, the message has been sent, not yet received, and is on the Bus; but in the first case, the message has been sent, not yet received, and is not on the Bus. It has disappeared from the system:

$$\exists \text{ NIC.Sent(msg,NIC)} \quad \text{AND NOT(Received(msg,IntendedReceiver(msg)))} \quad \text{AND NOT(On(msg,Bus))}$$

(Here I use the notation $\exists$x.A, with a ".", to say there is an x such that A.) This shows that we have already developed vocabulary sufficient to define Lost(msg). We may wish to introduce this term specifically via a meaning postulate:

$$\begin{array}{ll} & \text{Lost(msg)} \\ & \Leftrightarrow \\ \exists \text{ NIC.Sent(msg,NIC)} & \text{AND NOT(Received(msg,IntendedReceiver(msg)))} \\ & \text{AND NOT(On(msg,Bus))} \end{array}$$

The use of $\exists$ here is only nominal. We can eliminate its use when we consider

that there are only finitely many NICs, say NIC1, NIC2, . . ., NICk. Instead of writing ∃NIC.Sent(msg,NIC), we could instead have written

Sent(msg,NIC1) OR Sent(msg,NIC2) OR ..... OR Sent(msg,NICk)

Corrupted(msg) To talk about message corruption, it seems new vocabulary must be introduced, since it cannot easily be rephrased in the vocabulary we have. One way is to introduce a new predicate for it directly, as here. Another way may be to consider fundamental properties of messages, and rephrase Corrupted in terms of these fundamental properties. For example, it is very likely we shall want to speak during the refinement of the content of a message in terms of its fields and so forth. We could anticipate this by introducing a new function Content(msg). How might we want to express the content of a message logically: not its syntactic format (it might have many, say one On the Bus and another In a NIC), but its intended meaning? We have noted that an attribute-value-pair representation of messages can mitigate or eliminate comprehension issues following from a reordering of messages, so we might want to keep in mind that Content(msg) can be usefully expressed, when it comes to it, as a list of attribute-value pairs. At this Level, though, we have not considered what application the Bus is involved in, nor what kinds of messages are necessary for this application.

It seems apt to take Content as a term To Be Defined at a later Level. We can use it here to derive a meaning postulate for Corrupted, keeping in mind that the meaning postulate contains an undefined term which it is expected will be defined later. If we are willing to use the tense-logical operator $\Diamond_P$, we can say this: a message has been corrupted if its Content is not the same as it was sometime in the past:

$$\Diamond_P(\text{Sending(msg1,NIC1)} \quad \text{AND Content(msg1)} = \text{Y}) \\ \text{AND NOT (Content(msg1)} = \text{Y})$$

We can maybe make this a little slicker by introducing a function term for original content of a message, using the meaning postulate:

$$\text{OriginalContent(msg1)} = \text{Y} \\ \Leftrightarrow \\ \Diamond_P(\text{Sending(msg1,NIC1) AND Content(msg1)} = \text{Y})$$

Then to express message corruption all we need say is

NOT (OriginalContent(msg1) = Content(msg1))

We can now define Corrupted by means of a meaning postulate for it:

Corrupted(msg1) $\Leftrightarrow$ NOT (OriginalContent(msg1) = Content(msg1))

Phantom(msg) A phantom is a message that appears on the Bus, or is Received by a NIC, but was not Sent by anyone:

Phantom(msg)

$\Leftrightarrow$

(On(msg,Bus) OR Received(msg,NIC)) AND $\forall$ NIC.NOT Sent(msg,NIC)

Similarly to above, the use of $\forall$ here can be eliminated using NIC1, ..., NICk and AND.

InappropriateReceiver We have identified a HazHapp in

Received(msg,NIC) AND NOT (IntendedReceiver(msg) = NIC)

We can call the NIC in this case an inappropriate receiver, and introduce a meaning postulate for such a term:

InappropriateReceiver(msg,NIC1)

$\Leftrightarrow$

Received(msg,NIC1) AND NOT (IntendedReceiver(msg) = NIC1)

OutsideInterval Discussion reserved for a later Level when Interval(msg) is defined and OutsideInterval is defined on the basis of the definition of Interval.

PartlyOutsideInterval Discussion reserved for a later Level when Interval(msg) is defined and PartlyOutsideInterval is defined on the basis of the definition of Interval.

Partly-Attached Discussion reserved for the exercises for the reader.

Partly-Sending Discussion reserved for the exercises for the reader.

Partly-Receiving Discussion reserved for the exercises for the reader.

To summarise, the discussion has yielded meaning postulates for Lost, Corrupted, Phantom and InappropriateReceiver.

As shown in Figure 6.10, we have identified HazHapps for which new predicates were introduced but not defined, namely OutsideInterval(msg) and PartlyOutsideInterval(msg). We are not able at this Level to introduce meaning postulates to explicate the meaning of these terms, for they are intended to refer to a timing interval which

Predicate	Object Types	Status
Integrity	Bus	Physical
Integrity	NIC	Physical
Attached	NIC,Bus	Physical
AttachedToDevice	NIC	Physical
Sent	msg,NIC	Defined at Level 0
Received	msg,NIC	Defined at Level 0
Lost	msg	Defined at Level 0 using IntendedReceiver
Corrupted	msg	Defined at Level 0 using new function Content(msg) and/or defined function OriginalContent
Content	msg	To Be Defined
Phantom	msg	Defined at Level 0
IntendedReceiver	msg, NIC	To Be Defined
InappropriateReceiver	msg, NIC	Defined at Level 0 using IntendedReceiver
Sender	msg	Defined at Level 0
Interval	msg	To Be Defined
OutsideInterval	msg	To Be Defined
PartlyOutsideInterval	msg	To Be Defined
IntermittentlyAttached	NIC,Bus	To Be Defined and Discussed
CorruptedSending	msg,NIC	To Be Defined and Discussed
CorruptedReceiving	msg,NIC	To Be Defined and Discussed

Figure 6.11: Key Predicates Resulting from the HAZOP Analysis at Level 0

is part of the internal Content(msg), which is at this point an object with no defined internal structure. The definition of these terms must wait until a later refinement Level.

HazHapps have also possibly been identified associated with

- Partly-Attached(NIC)
- Partly-Sending(msg,NIC)

Term	Definition
Sent(msg,NIC)	NOT Sending(msg,NIC) AND $\Diamond_P$(Sending(msg,NIC))
Sender(msg1) = NIC1	Sent(msg1,NIC1)
Received(msg,NIC)	$\Diamond_P$(Receiving(msg,NIC)) AND NOT On(msg,Bus)
Lost(msg)	$\exists$ NIC.Sent(msg,NIC) AND NOT(Received(msg,IntendedReceiver(msg))) AND NOT(On(msg,Bus))
OriginalContent(msg1) = Y	$\Diamond_P$(Sending(msg1,NIC1) AND Content(msg1) = Y)
Corrupted(msg1)	NOT(OriginalContent(msg1) = Content(msg1))
Phantom(msg)	(On(msg,Bus) OR Received(msg,NIC)) AND $\forall$ NIC.NOT Sent(msg,NIC)
InappropriateReceiver(msg,NIC1)	Received(msg,NIC1) AND NOT (IntendedReceiver(msg) = NIC1)

Figure 6.12: Meaning Postulates Introduced at Level 0

Sending(msg,NIC) $\Rightarrow$ On(msg,Bus)
Receiving(msg,NIC) $\Rightarrow$ On(msg,Bus)
NOT AttachedToDevice(NIC) $\Rightarrow$ NOT Integrity(NIC)
NOT Attached(NIC,Bus) $\Rightarrow$ NOT Integrity(NIC)

Figure 6.13: Partial Meaning Postulates Introduced at Level 0

- Partly-Receiving(msg,NIC)

Discussion of these has been reserved for the exercises, but a few words here might be worthwhile. Partly-Attached, meant to refer to intermittent operation, can possibly be expressed using tense-logical operators: it was attached, then it wasn't, then attached again, then not, and so on. We might wish to distinguish this intermittent attachment from the case in which the NIC was once not attached, then attached again, and everything now works fine. This doesn't seem to be a distinction we can

Property	Object Types	Status
NOT Integrity	Bus	Primitive
NOT Integrity	NIC	Primitive
NOT AttachedToDevice	NIC	Primitive
NOT Attached	NIC,Bus	Primitive
Lost	msg	Defined at Level 0
Corrupted	msg	Defined via undefined term Content, to be defined at a later Level
Phantom	msg	Defined at Level 0
InappropriateReceiver	msg,NIC	Defined via undefined term IntendedReceiver, to be defined at a later Level
OutsideInterval	msg	To Be Defined
PartlyOutsideInterval	msg	To Be Defined
IntermittentlyAttached	NIC,Bus	Not Treated Yet
CorruptedSending	msg,NIC	Not Treated Yet
CorruptedReceiving	msg,NIC	Not Treated Yet

Figure 6.14: HazHapps Identified at Level 0 With Their Status

make with the current vocabulary. It may be worthwhile to introduce a new predicate IntermittentlyAttached(NIC) and to define it in later Levels of the refinement. Partly-Sending and Partly-Receiving seem to describe situations in which the message to be transmitted by the NIC is not the same as what goes on the Bus, respectively what was on the Bus is not the same as what is Received by the NIC. Again, these seem best definable at a Level at which we know in more detail how a NIC assembles data for sending or how it parses received data. We can introduce new terms

- IntermittentlyAttached(NIC)
- CorruptedSending(msg,NIC)
- CorruptedReceiving(msg,NIC)

whose definitions are to be given, maybe at Level 0, or maybe at later Levels of the refinement. Further consideration and discussion is left for the exercises. We

substitute these terms for Partly-Attached, Partly-Sending and Partly-Receiving in Figure 6.11 and subsequently.

6.4.8 Summary of Level 0

We have identified between ten and thirteen forms of HazHapp, depending on what we might make of Partly-Attached, Partly-Sending and Partly-Receiving. We have introduced some new terms (predicates and so on), listed in Figure 6.11, some defined at Level 0 and some requiring definition at later Levels. We are now finished with the first iteration of the CFA.

None so far

Figure 6.15: Assumptions on Which HazAn is Based, Level 0

In order to make the meaning postulates and define the vocabulary involved in identifying the HazHapps, as well as stating the assumptions, we introduced some further vocabulary that is not yet itself the subject of meaning postulates in Level 0, that must be defined by meaning postulates in later Levels. This vocabulary is listed in Figure 6.16. It consists entirely of predicates and relations, and no new objects, although it could profitably be considered whether Content is best considered as an object.

Content(msg,X)
IntendedReceiver(msg)
Interval(msg)
OutsideInterval(msg)
PartlyOutsideInterval(msg)
IntermittentlyAttached(NIC)
CorruptedSending(msg,NIC)
CorruptedReceiving(msg,NIC)

Figure 6.16: New Vocabulary to be Defined at Later Levels

6.4.9 Hazardous Factor Mitigation and Avoidance

At this Level, we have a vocabulary of objects (object types), their properties and relations, and we have identified within this vocabulary between ten and thirteen forms of HazHapp. Each of these phenomena must be addressed: either avoided or mitigated at Level 0, or to be addressed in further refinement levels.

One particular concept stands out. Corruption of msgs, along with CorruptedSending and CorruptedReceiving, can be standardly handled by using such well-known methods as checksums such as Cyclic Redundancy Codes (CRCs). Such use would avoid three of the thirteen HazHapps identified so far. It would also avoid the necessity of defining the new term Content.

6.5 Exercises

1. Investigate, define and justify the HazHapps associated with NOT Attached, NOT On, NOT In, NOT Sending and NOT Receiving.
2. Investigate, define and justify the HazHapps associated with Partly-Attached, Partly-On, Partly-In, Partly-Sending and Partly-Receiving.
3. Define what might be meant by IntermittentlyAttached(NIC). Try to formulate a formal definition in terms of Level 0 vocabulary and logic.
4. Define what might be meant by CorruptedSending(msg,NIC). Try to formulate a formal definition in terms of Level 0 vocabulary and logic.
5. Define what might be meant by CorruptedReceiving(msg,NIC). Try to formulate a formal definition in terms of Level 0 vocabulary and logic.

Bibliography

[1] Tim Bedford and Roger Cooke, *Probabilistic Risk Analysis: Foundations and Methods*. Cambridge University Press, 2001

[2] 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.

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

[4] Daniel M. Kammen and David M. Hassenzahl, *Should We Risk It?*, Princeton University Press, 1999.

[5] Hiromitsu Kumamoto and Ernest J. Henley, *Probabilistic Risk Assessment and Management for Scientists and Engineers*, Second Edition. IEEE Press, 1996.

[6] Peter Bernard Ladkin, *Causal System Analysis*, ebook, RVS Group, University of Bielefeld, 2001. Available at https://rvs-bi.de/publications/books/CausalSystemAnalysis/index.html , accessed 2016-07-26.

[7] Peter Bernard Ladkin, *An Overview of IEC 61508 on E/E/PE-System Functional Safety*, Causalis Limited, 2008. Available from https://causalis.com/90-publications/99-downloads/IEC61508FunctionalSafety.pdf , accessed 2017-07-31.

[8] 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.

[9] Leslie Lamport, *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems 16(3):872-923, May 1994.

[10] Bev Littlewood and Lorenzo Strigini, *Validation of ultrahigh dependability for software-based systems*, Comm. ACM 36(11):69-80, November 1993.

[11] David L. Parnas, *On the criteria to be used in decomposing systems into modules*, Comm. ACM 15(12):1053-1058, December 1972. Available from https://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf , accessed 2017-07-31.

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

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

[14] 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/