Ontological Hazard Analysis of a Communications Bus

Peter Bernard Ladkin
7 July 2010

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 [KamHas99]. A pressure vessel is also the first example in [KumHen96], 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 [BedCoo01], 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 [FTH81]). 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 $O(10^{-5})$ [LitStr93], 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 [ReChCa99] describes the adaptation of the process-industry method of Hazard and Operations Analysis, HAZOP, to the assessment of software.

Ontological Hazard Analysis

Here, we perform the beginnings of a Preliminary HazAn (PHA) on an example from digital electronics, a generic communications bus for road transport vehicles. We use Ontological Hazard Analysis (OHA), a technique devised by the author for PHA whereby the system is conceived at a high level of abstraction, at which the vocabulary within which system phenomena are to be described is made explicit and adhered to rigorously. The usual techniques of potential hazard elicitation may be applied (in the example here, HAZOP [ReChCa99]) to obtain a list of hazardous happenstances (HazHapps). These HazHapps obtain names in the vocabulary. Some of them 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. These are noted for working in further *refinement* steps. During the process, some convenient *assumptions* ultimately constraining the design of the artifact may be made, such as here that a data packet contains an integral number of fields, and fields are always of the form <attribute,value>. These constraints must be met at design time (by simulation if not hard-coded), otherwise the HazAn may be invalid.

At the end of an OHA stage, then, we have

- (a) an explicit vocabulary, the *refinement-stage primitives*
- (b) a list of hazardous happenstances (events or states), *HazHapps,* with associated *Safety Requirements* (*SafeReq*s: 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 HazAn,
- (d) *assumptions* made during the course of the analysis, typically about the form of data objects, which simplify the analysis,
- (e) introduced, as yet undefined, vocabulary (*new primitive* vocabulary) needed for expressing the HazHapps identified at this stage.

The **goal** during the stage will have been to be as complete as possible in compiling (b), in particular to the point of identifying SafeReqs; to get as many of the new primitives defined in terms of the stage primitives by meaning postulates as possible; and to keep the number of new primitives to a minimum.

When the stage is finished, a new refinement stage must be chosen, and then worked through similarly. A new refinement stage 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 stage, and expressing them 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 above, to identify SafeReqs 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 is simultaneously to reduce the new primitives as far as possible through deriving meaning postulates. When the OHA reaches a stage at which most of the identified HazHapps have associated SafeReqs, it can be terminated. Those HazHapps without associated SafeReqs must be retained and analysed at a later HazAn stage, 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 Preliminary HazAn.

OHA works on a hierarchy of abstractions, in the same way in which hierarchical decomposition is used in software design [Par72], 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 OHA 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 [Par72].

The formal application of these techniques in specification is known as *formal refinement.* Examples of rigorous formal refinement applied to specifications, in aid of the mathematical proof of correctness of certain algorithms, may be found in [Lad96], [Hen97], [LadLam99]. Formal refinement of this type goes way beyond the level of rigor we have found to be helpful in OHA.

OHA works, at any level of its hierarchy, with objects. Objects have properties, as well as relations

with each other. At a specific level, (types of) objects must be explicitly defined, as well as which properties of them will be considered and which relations between them. This constitutes the vocabulary for (a).

A summary of two OHAs, which were very different as they progressed, one using HAZOP and Causal System Analysis, the other using finite-automata formal refinement, may be found in [StSiLa09]. This paper recounts work published in detail in [Stu05], [Siek10].

A Generic Digital-Communication Bus

We work here through an example from programmable electronics, a generic digital-communication bus such as found in modern road transport vehicles, as illustrated in Figure 1. Sensors,which measure current values of such phenomena as force at wheels, rotational speed of wheels, position of steering frames, and braking, 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 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; also synchronising its internal clock with those of other NICs to 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.
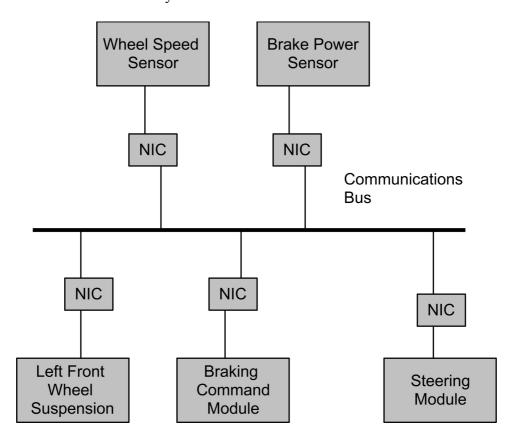

Figure 1: A Communications Bus for Road Vehicle Electronics

At this point, according to IEC 61508 (see [Lad08], Figure, p9, The IEC 61508 Life Cycle), 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.

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) [Lad01]. For the purpose of analysing how an accident can possibly result from system behavior, it will first be helpful not to distinguish: 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 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. We will refer to 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 one command-path, through messages in the communication bus, between brake command and actuation (that is, no alternative physical connection, say through a hydraulic-mechanical system), that the brakes will not be applied. I think as above that we can say that this represents a potential danger; 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, is 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. If we are designing a generic bus, along with generic NIC specifications, which is going to be used in varied applications by various manufacturers, we know 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 not necessarily of the precise waveform (for it usually is a continuous waveform) which is actually put on the medium by electronic means by the NICs. Further, we might know there are arbitration difficulties, such 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 [Dr03]. 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.

Level 0

We start typically with the highest level, which we call 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. In hierarchical design, a «natural number» may be defined in terms of an 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 proved – verified – that the operations on bits indeed implement the operations on integers at the «higher» level. Similarly, it must be verified in OHA 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; we will only be concerned with hazardous happenstance. This is the highest, most abstract, level, the natural place at which to start an OHA.

Indeed, we have so far considered messages very 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, 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 (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 later stages.

We start by defining 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.

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.

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 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)*.

Objects	Properties
Bus	*Integrity*
NIC	*Integrity*
msg	*Content*
	Size
	Deadline

Figure 2: Objects and Their Properties

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 follows, in which an "X" indicates that the relation has the object type as an argument:

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

Figure 3: Relations and Their Object Types

Figures 2 and 3 constitute the definition of the vocabulary for (a) of Level 0.

Meaning Postulates

I have noted above that the property of being *Sent* or begin *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):

$$Sending(msg,NIC) => On(msg,Bus)$$

$$Receiving(msg,NIC) => On(msg,Bus)$$

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:

$$Sent(msg,NIC) <=> NOT Sending(msg,NIC) AND Sometime-Past(Sending(msg,NIC))$$

$$Received(msg,NIC) <=> Sometime-Past(Receiving(msg,NIC)) AND NOT On(msg,Bus)$$

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

We must always remember, though, that the point of the current exercise is 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 than may strictly be needed to find out how things may go dangerously wrong.

Using HAZOP

We have a vocabulary for expression of some high-level communication-bus properties. We need to start identifying hazardous happenstances, HazHapps, in the operation of the bus, which are either describable with this vocabulary or with an extension of the vocabulary. We need to control any extension. We do so here by applying HAZOP to expressions in the primitive vocabulary.

HAZOP is a technique which, in its original form, depends crucially on group-think [ReChCa99]. We have found that, embedded in a refinement process, one person working alone, and 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, the HAZOP being used to just to provoke thought of hazards. If a hazard is missed at one stage, we have experienced that it is likely to turn up at a later stage. We have no theoretical grounds for this observation. The phenomenon does reduce the personpower resources needed to perform a satisfactory HazAn.

There are examples of OHA in which HAZOP turned out to be entirely superfluous to the HazAn, in which a demonstrably-complete set of Safereqs were derived at Level 0, and the formal refinement consisted of making the system more concrete (i.e., the usual goal of refinement) while preserving the SafeReqs through the refinement. Actual executable code was derived, in SPARK, and each refinement step was formally proved [Sie10].

In another example, a communications bus similar to this, the primitive vocabulary started rather large, and the extension of vocabulary through HAZOP in OHA was considerable [Stu05]. The extension was somewhat controlled by using causal system analysis of the indentified HazHapps with Extended Partial Why-Because Graphs (epWBG), which were then used to derive a Fault Tree for the system in a collaborative student project. The refinement had three stages, during each of which HAZOP and causal flow analysis with epWBGs was used. The Fault Tree contained about 170 event nodes. It is likely in this case that the HazAn would have benefitted from collaborative development, just as the Fault Tree conversion benefitted from the group work.

Interpreting HAZOP Guide Words for the Level 0 vocabulary

We recall that HAZOP uses the following guide words as hints towards system properties which might be hazardous or lead to hazardous happenstance:

No
More
Less
As well as

Part of
Reverse
Other than
Early
Late
Before
After
Faster
Slower
Where else

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 hazardous happenstance. Here, I state the results.

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 of further concern involving integrity.

Concerning *Content(msg)*, *Size(msg)* and *Deadline(msg)*, there is more to be said.

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 details of the msg, say, its ID, are present, but that the substantial content, that which is or will be important to the receiver, is not longer present.

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)*. We 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 appropriate 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 means that an extra message appears (*On* the *Bus*, or *Received* by

a *NIC*) that has not explicitly been generated – a phantom.

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. But to interpret this is a task for later, when the msg has been broken down into its component parts in the hierarchical decomposition. At most, what could be said at this level about this possibility is that certain timing fields do not contain the information with which they were written, and this is a form of msg *corruption*.

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:

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

To say this, we would need to introduce the new function *IntendedReceiver*.

In summary, consideration of the combination of HAZOP keywords with *Content* has led to the introduction of the new property, respectively function, *Corrupted(msg)* and *IntendedReceiver(msg)*. We may consider here, as a question of symmetry, whether we need a function name for the *Sender* of a message similar to *IntendedReceiver*. 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:

$$Sender(msg1) = NIC1 \quad <=> \quad Sent(msg1,NIC1)$$

showing that we have the appropriate vocabulary already to define the function *Sender*, which we did not have in the case of *IntendedReceiver*. Exactly how an *IntendedReceiver* is identified is a matter for later stages in the refinement.

Next we consider the combination of keywords with *Size(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)*. 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*.

Next, 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. 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 are taken here to be «primitive» and may become defined in later stages in which the length of the msg is taken into account.

Considering *Deadline* then, we have identified one more property (or function) of a message,

Interval(msg), and two assertions *OutsideInterval(msg)* and *PartlyOutsideInterval(msg)*, which can be involved in hazardous happenstance.

Considering the relations *Attached, On, In, Sending, Receiving*, I state without argumentation that the interpretable keywords are restricted to *Not* and *Partly. Not-Attached* and *Partly-Attached* may mean that the NIC is not attached, or 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.

Hazardous Happenstance: Summary and Discussion

The combinations of HAZOP keywords with the objects, properties and relations at the first level, combined with our preliminary considerations, have given us some insight into what can go hazardously wrong. We have identified the following hazardous happenstances.

NOT Integrity(Bus). This can be expressed, as here, using the usual logical operators on a property which is already in the vocabulary.

NOT Integrity(NIC). This can be expressed, as here, using the usual logical operators on a property which is already in the vocabulary.

?*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: 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:

Sent(msg) AND NOT(Received(msg,IntendedReceiver(msg))) AND NOT(On(msg,Bus))

showing that we have already developed vocabulary sufficient to define *Lost(msg)*. We may wish to introduce this term specifically via a meaning postulate:

Lost(msg)
<=>
Sent(msg) AND NOT(Received(msg,IntendedReceiver(msg))) AND NOT(On(msg,Bus))

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, what fields it has and so forth. We can anticipate this by introducing the function *Content(msg)*.

We can consider how we might want to express, logically, the content of a message: not its actual format on the Bus or in a NIC, but its intended meaning. We have noted above in the discussion of the application of keywords that an attribute-value-pair representation of messages can mitigate or eliminate comprehension issues following from a reordering of messages. So we can keep in mind that *Content(msg)* can be usefully expressed, when it comes to it, as a list of attribute-value pairs.

We may use the notion of *Content(msg)* to define when a message has been corrupted. Given that we have introduced the tense-logical operator *Sometime-Past*, we can surely say this: a message has been corrupted if its *Content* is not the same as it was sometime in the past:

Sometime-Past(Sending(msg1,NIC1) AND Content(msg1) = Y) and NOT(Content(msg1) = Y)

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

OriginalContent(msg1) = Y
<=>
Sometime-Past(Sending(msg1,NIC1) AND Content(msg1) = Y)

Then to express message corruption all we need say is

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

And we can introduce the vocabulary *Corrupted* by means of a meaning postulate for it:

Corrupted(msg1) <=> NOT(OriginalContent(msg1) = Content(msg1))

Moving on, we have identified possibly hazardous happenstance in

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

We can call *NIC1* in this case an inappropriate receiver, and introduce a meaning postulate for such a term:

InappropriateReceiver(msg,NIC1)
<=>
Received(msg,NIC1) AND NOT (IntendedReceiver(msg) = NIC1)

We have identified possibly hazardous happenstance, for which we have introduced new predicates, in

OutsideInterval(msg)
PartlyOutsideInterval(msg)

but we are not able at this stage to introduce meaning postulates or other devices to explicate the meaning of these terms, for the reason that they are intended to refer to a timing interval which is part of the internal *Content(msg)* which is for us at this point an atom, an object with no defined internal structure. The definition of these terms must wait until a later refinement stage.

We have identified hazardous happenstance in

NOT Attached(NIC)
Partly-Attached(NIC)
Partly-Sending(msg,NIC)
Partly-Receiving(msg,NIC)

The first of these, *NOT Attached(NIC)*, is self-explanatory, as well as expressable in the vocabulary we already have. The second, 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. The problem we would have is 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 make with the current vocabulary. Best is maybe to introduce a new predicate

$$IntermittentlyAttached(NIC)$$

and to postpone its definition until later stages of the refinement. Concerning *Partly-Sending* and *Partly-Receiving*, these 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 stage 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

$$CorruptedSending(msg,NIC)$$
$$CorruptedReceiving(msg,NIC)$$

whose definitions are to be given at later stages of the refinement.

Hazardous Happenstance: Final Determination and Extended Vocabulary

We have the hazardous happenstances defined in terms of existing vocabulary

$$NOT\ Integrity(Bus)$$
$$NOT\ Integrity(NIC)$$

as well as the hazardous happenstances defined in terms of meaning postulates

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

and the new, as yet undefined, terms for hazardous happenstances that must be defined at later stages in the refinement:

$$OutsideInterval(msg)$$
$$PartlyOutsideInterval(msg)$$
$$IntermittentlyAttached(NIC)$$
$$CorruptedSending(msg,NIC)$$
$$CorruptedReceiving(msg,NIC)$$

So we have identified eight forms of hazardous happenstance. In order to reach this determination, we have introduced new functional terms

$$Sender(msg)$$
$$IntendedReceiver(msg)$$

the former with a meaning postulate, and the latter to be defined at a later stage in the refinement. We have also introduced new relations with the help of tense logic:

$$Sent(msg,NIC)$$
$$Received(msg,NIC)$$

This new vocabulary, four types of terms, is added to the existing vocabulary *Bus*, *NIC*, *msg*,

Integrity (twice: of *Bus* and of a *NIC*), *Content*, *Size*, *Deadline* as we proceed to further stages of the refinement. We are now finished with the first iteration of the OHA.

We summarise the results, in the form of the lists (b) – (e), for Level 1. There are no new object types identified at Level 1, so we omit List (a).

HazHapp	Safety Requirement
NOT Integrity(Bus)	Not yet identified
NOT Integrity(NIC)	Not yet identified
Lost(msg)	Not yet identified
Corrupted(msg)	Not yet identified
InappropriateReceiver(msg,NIC)	Not yet identified
OutsideInterval(msg)	Not yet identified
PartlyOutsideInterval(msg)	Not yet identified
IntermittentlyAttached(NIC)	Not yet identified
CorruptedSending(msg,NIC)	Not yet identified
CorruptedReceiving(msg,NIC)	Not yet identified

Figure 4: List (b) for Level 0, HazHapps Identified and Associated SafeReqs

In order to reach these HazHapps with as-yet-undefined Safety Requirements, we defined a number of meaning postulates, List (c), as well as made some assumptions which must be carried as assumptions through further levels until they may be made concrete during design, List (d). These are as follows.

Sending(msg,NIC) => On(msg,Bus)
Receiving(msg,NIC) => On(msg,Bus)
Sent(msg,NIC) <=> NOT Sending(msg,NIC) AND Sometime-Past(Sending(msg,NIC))
Received(msg,NIC) <=> Sometime-Past(Receiving(msg,NIC)) AND NOT On(msg,Bus)
Sender(msg1) = NIC1   <=>   Sent(msg1,NIC1)
Lost(msg) *<=>* *Sent(msg) AND NOT(Received(msg,IntendedReceiver(msg))) AND NOT(On(msg,Bus))*
OriginalContent(msg1) = Y *<=>* *Sometime-Past(Sending(msg1,NIC1) AND Content(msg1) = Y)*
Corrupted(msg1) <=> NOT(OriginalContent(msg1) = Content(msg1))
InappropriateReceiver(msg,NIC1) *<=>* *Received(msg,NIC1) AND NOT (IntendedReceiver(msg) = NIC1)*

Figure 5: List (c) for Level 0, Meaning Postulates

None so far

Figure 6: List (d) for Level 0, Assumptions Upon Which the HazAn is Based

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; that is, *primitive* vocabulary that must be defined by meaning postulates in later stages, in lower Levels of the refinement (or in the design). This vocabulary is as follows.

Sender(msg)
IntendedReceiver(msg)
CorruptedSending(msg,NIC)
CorruptedReceiving(msg,NIC)

Figure 7: List (e) for Level 0, New Vocabulary to be Defined at Later Stages

Hazardous Factor Mitigation and Avoidance

At this stage, we have a vocabulary of objects (object types), their properties and relations, and we have identified within this vocabulary eight forms of hazardous happenstance(HazHapp). Each of these hazards must be addressed: either avoided or mitigated at this very level, or retained as to-be-addressed in further refinement levels. At this point, we can perform no effective mitigation or avoidance, because each of the HazHapps is an atom (a non-structured situation) or a Boolean construct of atoms. We need to pass to the next refinement level in order to start addressing HazHapp mitigation or avoidance.

Level 1: The First Refinement Level

We select the HazHapp *Lost(msg)* as the driver of the refinement. We need to add structure to messages in order to say what it means for a message to be lost, and maybe to perform a causal system analysis of lost messages; to identify all of the causal factors which lead or may lead to a lost message, as well as their causal interrelations.

After we have performed the causal system analysis, we may be able to assign likelihoods to causal factors which appear in the analysis, and may be able to combine those likelihoods to estimate a likelihood of a message being lost. The acceptable likelihood of this HazHapp will generally be derived from the «acceptable risk» determined in a IEC 61508-conformant process.

Moving to Level 1: Structuring Messages and Message-Passing

At this Refinement Level 1, we could assume a bus protocol which does not rely on *NIC*s synchronising clocks, but is *opportunistic*, or event-triggered, as in Ethernet. An opportunistic bus protocol works roughly as follows. A *NIC*, say *NIC1*, which wants to transmit *msg1*, first tests the *Bus* to see if there is a message already on it. If not, *NIC1* may begin to transmit. It may be that another NIC, say *NIC2*, has already started transmitting *msg2*, but that *msg2*'s wave front has not yet reached the *Bus* connector of *NIC1* and so was not sensed as *NIC1* tested the *Bus*. *NIC1* starts transmitting *msg1*, and the wave front of *msg1* and *msg2* will subsequently collide somewhere. The result of two colliding messages we may take always to be nonsense. So such an opportunistic bus

protocol must define a method of collision detection, collision communication (certainly to the senders and receivers of the colliding messages, maybe to every NIC), and resolution (which in the case of Ethernet has *NIC1* and *NIC2* retrying, after waiting distinct or likely-distinct periods of time).

An alternative transmission protocol which we could consider, called a *time-triggered* or *round-robin* protocol, requires the NICs to synchronise their internal clocks. They can do this, and maintain synchronisation, within certain error bounds using known and well-analysed algorithms. Time slots are defined by the protocol, one slot for each possible transmission. So, if one had 40 NICs, one could define a cycle size of 40 time slots, and *NIC-k* would be able to put its message on the bus in the k'th time slot of each cycle.

Round-robin scheduling has no collisions, by construction, so no need for collision detection and resolution, provided that the clock synchonisation is reliable (the boundaries of adjacent time slots may be separated from each other by a period equal to or greater than the error bound). However, there may be NICs that need to transmit only rarely, and other NICs which transmit in every cycle. Most of the slots of those NICS which transmit rarely will not be used. Thus there can arise a problem of efficiency: a NIC which needs to transmit must wait for its slot, and in the meantime a lot of free capacity flows by as it is waiting.

This is not the place to discuss thoroughly the merits and demerits of various bus protocols, but we do need this much of an overview in order to specify the vocabulary which we introduce at Level 1.

Level 1 General Definitions and General Meaning Postulates

Time-triggered or event-triggered, messages are usually decomposed into packets. Packet-oriented protocols work best when packets are of more or less uniform size, so let us assume this will be a criterion in devising the decomposition of a message into packets. We shall need a new relation, saying that a packet constitutes part of a message. We rearrange the syntax to bring the predicate to the front, followed by its subject and object, as usual:

ConstitutesPartOf(pkt,msg)

A packet may be an arbitrary division of a message, such as the pages of a book are an arbitrary division of its paragraphs, but we may on the other hand insist that a packet is a meaningful part of a message; that is, that it contains specific fields, and all the information in each field, of the message. If the fields are expressed as <attribute,value> pairs, that means that a packet contains whole <attribute,value> pairs; none of them is split across more than one packet. If we wish, we may introduce a predicate to say that a specific field, which we may consider abstractly as an <attribute,value> pair, is contained in a specific packet:

FieldContainedIn(field,pkt)

which will have binary values *True* or *False*, and not a third value *Partly-True, Partly-Somewhere-Else*. We may also be specific as to what a field is:

Field(x) <=> there is an *attribute*, and there is a *value*, such that *x = <attribute,value>*

If we wish to go this route, we can define all manner of *attributes* and *values* during refinement. At Level 1, before we have developed these specifics, *attribute* and *value* could become new object types, but we may equally well choose to leave this to a later refinement step, and do so.

It is usual for information that is required to be reliable to include some reliability checks, for example a simple parity check, or some more complex CRC calculation. We can refer to such a calculation generically as a *Checksum*. So a packet can be considered to have a C*hecksum*, new vocabulary introduced now, and this C*hecksum* will be *valid* or *invalid*, a firm binary choice, leading to the formulation that *Checksum be* a function with defined binary values *valid, invalid*. Later on in the development, when *Checksum* has been implemented as a specific function, it may be determined from empirical knowledge of the algorithm used to implement *Checksum* what the chances are that a given *Checksum* will be invalid. This may be used in a risk calculation.

We have said that packets should have a more or less uniform size. This suggestion comes from decades of experience with computer networks, and it seems wise to take it over. Since a message consists of an integral number of packets, and the packets have more or less uniform size, it seems reasonable to take the *Size* of a *msg* to consist of the number of packets which constitute it:

$$Size(msg) = \# \{pkt \mid ConstitutesPartOf(pkt,msg)\}$$

This of course is a meaning postulate for the predicate *Size* in terms of the new objects (object types) defined at Level 1.

We can also say something more now, informally, about what a NIC must do. At the original level, a NIC was an atom. Here we can say that, in its *Sender* role, it has the task of decomposing a *msg* into *pkt*s, or maybe composing all the *<attribute,value>* pairs which are to go into a specific *msg* into *pkt*s of more or less uniform size, and then putting those *pkt*s on the *Bus* in some random or meaningful order. And in its *Receiver* role, it has the task of *Receiving* all the *pkt*s comprising a particular *msg* and sending all the *<attribute,value>* pairs comprising that *msg* onwards to their intended destinations. We could pack all this information about the actual tasks of a *NIC* into Level 1, along with the decomposition of *msg*s into *pkt*s, but experience with refinement has shown that it is preferable to perform one refinement task at once, and leave others to a sequential refinement step. So we keep in mind here what the decomposition of a *msg* into *pkt*s entails for the function of a *NIC*, and resolve to refine a *NIC* in a further refinement step, but not in this one.

I note here that certain protocols, such as the Internet protocol suite TCP/IP, take different design decisions from those we here take. TCP/IP divides data into packets, for example, without paying attention to where the packet boundary occurs with respect to meaningful content (fields). TCP/IP just chops data up, as the pages of a book chop up the prose of the book's author. TCP relays the packet with a sequence number, equivalent to the page of a book, which is used to reassemble the entire data stream from its packets at the destination (indeed, this is a major function of the receiving code of TCP). The representation of fields as *<attribute,value>* pairs, and the requirement that <attribute,value> pairs are contained within precisely one packet and not split across packets, obviates the need for sequence numbering for message-reassembly. The meaningful contents of a *msg* are just the collection of the meaningful contents of its *pkt*s; order is not relevant.

This observation enables us to derive another meaning postulate, this time for *Received*. A *msg* has been *Received* by a *NIC* just in case all its *<attribute,value>* pairs have been read by the *NIC*, which is the case precisely when all *pkt*s constituting the *msg* have been read by the *NIC*, and the *NIC* has composed all *pkt*s back into the *msg* of which they constitute part:

$$Received(msg,IntendedReceiver(msg))$$
$$<=>$$
$$For\ all\ pkt\ [\ (ConstitutesPartOf(pkt,msg) =>$$
$$(Received(pkt,IntendedReceiver(msg))\ AND\ Composed(msg,IntendedReceiver(msg))\ ]$$

Level 1 Hazard Analysis for *Lost(Msg)*

The structure and vocabulary we have introduced so far allowed us to introduce a meaning postulate for the HazHapp *Lost(msg)*, which is, we recall, that *Lost(msg)* is equivalent to

Sent(msg) AND NOT(Received(msg,IntendedReceiver(msg))) AND NOT(On(msg,Bus))

This constrains in useful ways our considerations as to what constitutes losing a *msg*. Let me illustrate.

What are the ways in which a *pkt* could not successfully reach its destination? Let us consider the temporal progress of a pkt; it will get pulled up short somewhere. First, suppose the *pkt* is not put on the *Bus* by its *NIC*. It was supposed to be put on the *Bus*, but it wasn't.. We have some of the vocabulary to say this already, but not yet which NIC a pkt belongs to. Let us remedy this, also with reception:

Sender(pkt) = Sender(msg) where *ConstitutesPartOf(pkt,msg)*
IntendedReceiver(pkt) = IntendedReceiver(msg) where *ConstitutesPartOf(pkt,msg)*

We have one criterion, the earliest, for non-reception of a *msg*: the *pkt*s weren't put on the *Bus* by their *NIC*. But now look at the meaning postulate for *Lost*: it contains a conjunct which says the *msg* was *Sent* ! We define here the Sending of a *msg* by the meaning postulate

Sent (msg) <=> ForAll pkt (ConstitutesPartOf(pkt,msg) => Sent(pkt))

So if a *pkt* doesn't make it onto the *Bus* from its *NIC*, this does not constitute part of its associated *msg* being *Lost.* But we don't want this failure mode just to evaporate out of our considerations: do we have a HazHapp, part of which it does constitute? Yes, we do. Putting *pkt*s correctly on the bus has to do with the HW or SW *Integrity* of the *NIC*, and we have already identified *NOT Integrity(NIC)* as a HazHapp at the first level. It is covered: we will deal with it under that rubric.

So the *msg* was put on the *Bus*: all of it, all its *pkt*s. But it's not there now, and it wasn't *Received* by its *IntendedReceiver.* So if it's not *On* the *Bus* now, that means either the time has gone by which the waves of the *pkt*s will have taken to reach *IntendedReceiver*, or one is still within the time within which some *pkt* should still be in transit – but it's not. Than can only happen in two ways. (1) If the bus has been physically compromised, broken or at least damaged enough to attentuate the signal so that it is indistinguishable from background, then the *pkt* will "disappear" in this sense. (2) If there is a collision, and this collision goes undetected, then the *pkt* as information will have disappeared; its contents will have been incorporated into the noise resulting from the collision. Event type (1) is encapsulated in the HazHapp *NOT Integrity(Bus)*, so we will have this case covered when we perform the hazan for *NOT Integrity(Bus)*. Event type (2) remains a possibility not (yet?) covered elsewhere. Let us denote it by the here-unanalysed primitive *UndetectedCollision(pkt)*.

So, the situations other than *UndetectedCollision(pkt)* with which we are concerned here have been reduced to the case in which the *pkt*s reach their destination *IntendedReceiver(pkt) = IntendedReceiver(msg)* , but in which

NOT Received(msg,IntendedReceiver(msg))

A *msg* is *Corrupted* if some, but not all, of the *pkt* successfully reach their destination *IntendedReceiver(msg)*, or if some of the <attribute,value> pairs in msg become altered. So we may usefully deal here with one of the cases of *Corrupted(msg)* as well.

If a *pkt* reaches its destination *IntendedReceiver(pkt)* but is not *Received*, what could be the reason? Either the *pkt* is faulty or the *NIC* that is the *IntendedReceiver* is faulty. Again, this latter falls under the already listed HazHapp *NOT Integrity(IntendedReceiver(pkt))* so the case with which we are finally concerned is that the *pkt* is faulty. There are two ways in which a *pkt* can be faulty, namely (a) that its *Checksum* is *Invalid*, or that one or more of its *<attribute,value>* pairs has been modified, along with the *Checksum*, so that a different message is read - and thought valid! - than was sent. We may choose methods to implement *Checksum* in a later refinement step to make the chances of this as small as we like. What we should do here is note that it can theoretically occur, and mark it as a HazHapp.

$$Modified(pkt) <=> ....$$

We note that a modified *pkt*, because its *Checksum* is valid, will be *Received* by an *IntendedReceiver* which retains its *Integrity*. So a *Modified(pkt)* does not contributed to a *Lost(msg)*, as *Lost* is defined above. Therefore we are left only with the following interpretation of the part of the HazHapp associated with *Lost* and *Corrupted* that is not already covered by other listed hapzhapps

$$ThereIsSome\ pkt\ (ConstitutesPartOf(pkt,msg)\ AND\ Checksum(pkt) = Invalid)$$

Level 1 Rearrangement of HazHapps

We have seen that there is some commonality between *Lost* and part of *Corrupted*, namely pkts whose Checksum is invalid, and there is another part of *Corrupted* which concerns a different phenomenon, in which <attribute,value> pairs have been modified, and the Checksum accordingly.

The core HazHapp identified at Level 1 that characterises the first phenomenon we may call

$$VisiblyCorrupted(msg)$$
$$<=>$$
$$ThereIsSome\ pkt\ (ConstitutesPartOf(pkt,msg)\ AND\ Checksum(pkt) = Invalid)$$

The second phenomenon we have already denoted above by the predicate *Modified*.

We may thus remove *Lost(msg)* and *Corrupted(msg)* from our list of HazHapps and replace them by *VisiblyCorrupted(msg)*, because they are both definable in terms of *VisiblyCorrupted*. The progress in so doing arises from the ease with which mitigation or avoidance procedures may be conceived, and the formulation of safety requirements which involve such procedures.

Level 1 HazHapp Avoidance and Mitigation

Procedures with which to deal with an occurrence of *VisiblyCorrupted(msg)*, that is, a message which contains visibly corrupted packets, are ubiquitous in network protocol engineering. One of the most well-known, also used in the Internet reliable-transmission protocol TCP, is called the Sliding Windows protocol. All relevant features of Sliding Windows are known, including reliability rates for various versions. Here is not the place to discuss it; there are many texts. We refer for example to [Tan03].

Procedures to deal with Checksums not correctly identifying *Modified* packets may be found, along with reliability estimates, in textbooks on error-detecting and error-correcting codes, for example [LiCo04].

Thus at Level 1 we have categorised four major classes of HazHapps, *Lost(msg)*, *Corrupted(msg)*, *Modified*(msg), and *UndetectedCollision(pkt)*. We performed a small vocabulary change to unify phenomena associated with the first two, introducing the predicate *VisiblyCorrupted*.

We have identified known avoidance and mitigation methods in the literature for the phenomena subsumed under *Modified* and *VisiblyCorrupted*. To rule out the HazHapps *Modified* and *VisiblyCorrupted*, we may designate these methods as Safety Requirements (SafeReqs). The SafeReqs are listed in a table, and this list is carried through all refinement steps and into the design and implementation stages.

HazHapp	Safety Requirement
VisiblyCorrupted(pkt)	Reliable Transmission, e.g. Sliding Windows
UndetectedCollision(pkt)	Reliable Collision Detection, e.g. as in Ethernet
Modified(pkt)	Not yet identified

Figure 8: Partial List (b) for Level 1, HazHapps and Ensuing Safety Requirements

HazHapp (informal description)	Hazard Class
pkt not successfully put on *Bus* by *NIC*	*NOT Integrity(NIC)*
pkt lost in transit without collision	*NOT Integrity(Bus)*

Figure 9: Partial List (b) for Level 1, Identified HazHapps Retained for Processing Later

Summary of Level 1 Results

A key feature of OHA is the careful control of the expression of system characteristics, including hazards and failures, by using a controlled vocabulary along with features of logical languages. A second key feature is the use of refinement, to provide more detail about the system and simulatneously extend the vocabulary, guiding this refinement by the classification of hazards, and steps to identify and catalog mitigation and avoidance.

We started by considering a generic communications bus for a road vehicle which uses this bus for control and sensorics, and the hazards that could arise through use of this bus for these purposes. We identified at Level 0 necessary vocabulary with which to talk about these hazards, and we identified hazards, but could not analyse them, since they appeared at Level 0 as primitive vocabulary.

In OHA, it is recommended not to bite off too much at once: not to attempt to introduce in one step new concepts and vocabulary to handle all HazHapps identified at Level 0. We thus proceed to Level 1 by picking one HazHapp and analysing it further.

At Level 1, we considered in detail the HazHapp *Lost(msg)*. We were able to classify some of the phenomena which might lead to a lost message under other HazHapps, namely *NOT Integrity(NIC)* and *NOT Integrity(Bus)*. One part of the intuitive phenomenon associated with *Lost(msg)* that was not assimilated to these other HazHapps turned out to have much in common with the phenomena associated with another HazHapp *Corrupted(msg)*, and so we extended consideration to the similar

characteristics involved also in *Corrupted(msg)*. The remaining part of the phenomenon of *Corrupted(msg)* was identified through introducing a new property.

The vocabulary was realigned to relinquish use of *Lost* and *Corrupted* in favor of the use of *VisiblyCorrupted* and *Modified*, which predicates align exactly with known mitigation and avoidance procedures with well-known reliability characteristics, of which use can be made in the risk analysis which will follow the hazard analysis.

At this stage, we have identified the following HazHapps which have not been handled at Level 1 and therefore will become the subject of further refinement steps:

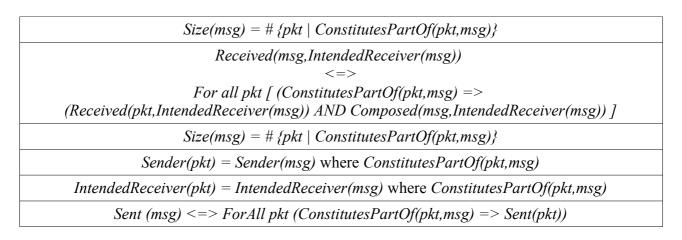| Unanalysed HazHapps for Further Refinement After Level 1 |
|---|
| |
| *NOT Integrity(Bus)* |
| *NOT Integrity(NIC)* |
| *InappropriateReceiver(msg,NIC)* |
| *OutsideInterval(msg)* |
| *PartlyOutsideInterval(msg)* |
| *IntermittentlyAttached(NIC)* |
| *CorruptedSending(msg,NIC)* |
| *CorruptedReceiving(msg,NIC)* |
| *UndetectedCollision(pkt)* |

Figure 10: Partial List (b) for Level 1, Already-identified HazHapps to be Analysed

We list the new object types, meaning postulates, assumptions, and new primitives at Level 1.

Object	Intended Interpretation
field	<attribute,value> pair

Figure 11: List (a) for Level 1, New Object Types Introduced

Since we have a new object type, *field*, we can renounce the meaning postulate for *Field(x)*.

Size(msg) = # {pkt | ConstitutesPartOf(pkt,msg)}
Received(msg,IntendedReceiver(msg)) *<=>* *For all pkt [(ConstitutesPartOf(pkt,msg) =>* *(Received(pkt,IntendedReceiver(msg)) AND Composed(msg,IntendedReceiver(msg))]*
Size(msg) = # {pkt | ConstitutesPartOf(pkt,msg)}
Sender(pkt) = Sender(msg) where ConstitutesPartOf(pkt,msg)
IntendedReceiver(pkt) = IntendedReceiver(msg) where ConstitutesPartOf(pkt,msg)
Sent (msg) <=> ForAll pkt (ConstitutesPartOf(pkt,msg) => Sent(pkt))

Modified(pkt) <=> ....

VisiblyCorrupted(msg)   *<=>*   *IsSome pkt (ConstitutesPartOf(pkt,msg) AND Checksum(pkt) = Invalid)*

Figure 12: List (c) for Level 1, Meaning Postulates

A *pkt* consists of an integral number of *field*s

Figure 13: List (d) of assumptions Introduced at Level 1

ConstitutesPartOf(pkt,msg)
FieldContainedIn(field,pkt)
IntendedReceiver(msg)
UndetectedCollision(pkt)

Figure 14: List (e) of New Primitives Introduced at Level 1

Refinement continues. To identify the next Level (Level 2), we consider the HazHapps in Figure 6, and use these as guidance.

Level 2 Refinement

There is one HazHapp in Figure 6 identified with the *Bus*, namely *NOT Integrity(Bus)*. We have already considered the possibility of undetected collisions, and identified a safety requirement to mitigate those, so the *pkt* losses associated with *NOT Integrity(Bus)* for which safety requirements remain to be formulated are those associated with physical-electrical failures such as short-circuits and breaks in the cabling. These can also be handled through known techniques from electrical engineering. They cannot be ruled out, because physical activities such as someone deliberately slicing a cable cannot be ruled out, but they can be detected. Considering NOT Integrity(BUS) thus leads us to the following Level 2 refinement.

New Objects	New Properties	New Relations
None	None	None

Figure 15: List (a) for Level 2, New Entities Added

HazHapp	Safety Requirement
NOT Integrity(Bus)	Electrical-anomaly detection, e.g., arc-fault circuit interrupters

Figure 16: Partial List (b) for Level 2, HazHapps and Ensuing Safety Requirements

Unanalysed HazHapps for Further Refinement After Level 2
NOT Integrity(NIC)
InappropriateReceiver(msg,NIC)
OutsideInterval(msg)
PartlyOutsideInterval(msg)
IntermittentlyAttached(NIC)
UndetectedCollision(pkt)
CorruptedSending(msg,NIC)
CorruptedReceiving(msg,NIC)

Figure 17: Partial List (b) for Level 2, Already-identified HazHapps to be Analysed Later

Deciding on Level 3

There are only two sorts of objects, *msg* and *NIC*, remaining in the cumulative list of HappHaps to be analysed and mitigated. Of these, there are HazHapps associated with *NIC* or *msg* with *NIC*, and there are two HazHapps associated purely with *msg*. The HazHapps *OutsideInterval* and *PartlyOutsideInterval* refer to deadlines and timing of messages. The other HazHapps refer either to a *NIC* alone, or to processing of a *msg* through a *NIC*. Since a *msg* is a data object, and not an active agent (it does not execute actions), we can usefully regard *CorruptedSending* and *CorruptedReceiving* as hazards associated with unsuccessful actions of the *NIC*, and therefore reflecting on the *Integrity* of the *NIC*, which in general terms means the ability of the *NIC* to carry out its required functions in an appropriate time. There are two ways *Integrity* might therefore fail. One way is that a required processing function is not carried out. The other way is that appropriate timing is not achieved. That appropriate timing constraints are not adhered to also reflects on *msg* deadlines, which in turn are associated so far with *OutsideInterval(msg)* and *PartlyOutsideInterval(msg)*. In the other direction, that a *msg* does not encounter its receiving *NIC* in time does not necessarily reflect on the *Integrity* of the receiving *NIC*. So there appears to be an asymmetry, in that considering the *Integrity* of a *NIC* may well lead to modification of the HazHapps designated as *OutsideInterval* and *PartlyOutsideInterval*, but considering these latter will not reflect in any particular way on the *Integrity* of the receiving *NIC*.

These reflections point us firmly in the direction of considering *Integrity(NIC)*, and refining the required operations of the *NIC* at the next refinement level, Level 3. Considering *msg* timing constraints will occur, then, at a further level of refinement beyond this.

The exact functioning of the NICs has at this point not been addressed at all. Nothing has been said about what a NIC shall do. Indeed, nothing has been said about the protocol under which the Bus will run and which parts of the protocol will be the responsibility of the NICs to ensure. We can observe here that, if the bus runs under a purely event-driven protocol, such as Ethernet, timing constraints on message reception are driven purely by the timing constraints of the attached equipment, whereas if the protocol is partly or completely time-triggered, there are possibilities for a message to arrive out-of-slot or partly-out-of-slot, thereby a HazHapp, which phenomena are purely internal to the communications and do not have to do with constraints of the attached equipment.

We shall leave the HazAn at Level 3 and beyond to the reader.

Overall Summary

We have illustrated here Ontological Hazard Analysis, which proceeds by *controlling the vocabulary in which system properties can be expressed*, starting at a very high level with few *object types, properties and relations* and extending the vocabulary only by necessity as HazHapps are identified which need to be expressed. Extending the vocabulary, and the expression within that vocabulary, gradually is a process known in the formal methods community as *formal refinement*. The method we have used here to identify HazHapps is HAZOP [op. cit.]. The use of formal refinement enables cumulative tables to be built of hazards identified Level by Level, and mitigating methods to be applied in the implementation, assuring coverage of as many hazards as can be expressed using the controlled vocabulary, and bringing the advantages of hierarchical development [op. cit.] to hazard analysis.

Other methods than those illustrated here may be used in OHA. For a very similar example, in which a different starting vocabulary was used, along with HAZOP, and further with Causal Flow Analysis using Extended Partial Why-Because Graphs (epWBAs), see [Stu05].

We summarise finally the HazHapps which have been dealt with already, along with the safety requirements that were identified as mitigation.

HazHapp	Safety Requirement
VisiblyCorrupted(pkt)	Reliable Transmission, e.g. Sliding Windows
UndetectedCollision(pkt)	Reliable Collision Detection, e.g. as in Ethernet
Modified(pkt)	Not yet identified
NOT Integrity(Bus)	Electrical-anomaly detection, e.g., arc-fault circuit interrupters

Figure 18: Partial List (b), HazHapps With Ensuing Safety Requirements at Levels up to and including Level 2

Unanalysed HazHapps for Further Refinement After Level 2
NOT Integrity(NIC)
InappropriateReceiver(msg,NIC)
OutsideInterval(msg)
PartlyOutsideInterval(msg)
IntermittentlyAttached(NIC)
UndetectedCollision(pkt)
CorruptedSending(msg,NIC)
CorruptedReceiving(msg,NIC)

Figure 19: Partial List (b) up to and including Level 2, HazHapps to be Analysed Later

In order to reach these HazHapps with Safety Requirements, we defined a number of meaning postulates, as well as made some assumptions which must be carried as assumptions through further levels until they may be "cashed out", made concrete. These are as follows.

Sending(msg,NIC) => On(msg,Bus)
Receiving(msg,NIC) => On(msg,Bus)
Sent(msg,NIC) <=> NOT Sending(msg,NIC) AND Sometime-Past(Sending(msg,NIC))
Sent (msg) <=> ForAll pkt (ConstitutesPartOf(pkt,msg) => Sent(pkt))
Received(msg,NIC) <=> Sometime-Past(Receiving(msg,NIC)) AND NOT On(msg,Bus)
Received(msg,IntendedReceiver(msg))   *<=>*   *For all pkt [(ConstitutesPartOf(pkt,msg) =>*   *(Received(pkt,IntendedReceiver(msg)) AND Composed(msg,IntendedReceiver(msg))]*
Field(x) <=> there is an attribute, and there is a value, such that x = <attribute,value>
*Size(msg) = # {pkt	ConstitutesPartOf(pkt,msg)}*
Sender(msg1) = NIC1   <=>   Sent(msg1,NIC1)
Lost(msg)   *<=>*   *Sent(msg) AND NOT(Received(msg,IntendedReceiver(msg))) AND NOT(On(msg,Bus))*
OriginalContent(msg1) = Y   *<=>*   *Sometime-Past(Sending(msg1,NIC1) AND Content(msg1) = Y)*
InappropriateReceiver(msg,NIC1)   *<=>*   *Received(msg,NIC1) AND NOT (IntendedReceiver(msg) = NIC1)*
Modified(pkt) <=> ....
VisiblyCorrupted(msg)   *<=>*   *IsSome pkt (ConstitutesPartOf(pkt,msg) AND Checksum(pkt) = Invalid)*
Sender(pkt) = Sender(msg) where ConstitutesPartOf(pkt,msg)
IntendedReceiver(pkt) = IntendedReceiver(msg) where ConstitutesPartOf(pkt,msg)

Figure 20: List (c) for Levels up to and Including Level 2, Meaning Postulates

A *field* consists in an *<attribute,value>* pair
A *pkt* contains an integral number of *Field*s
A *msg* consists of an integral number of *pkt*s

Figure 21: List (d) for Levels up to and Including Level 2, Assumptions

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; that is, *primitive* vocabulary that must be defined by meaning postulates in later stages, in lower Levels of the refinement (or in the design). This

vocabulary is as follows.

Sender(pkt); Sender(msg)
IntendedReceiver(pkt); *IntendedReceiver(msg)*
CorruptedSending(msg,NIC)
CorruptedReceiving(msg,NIC)
ConstitutesPartOf(pkt,msg)
Composed(msg,IntendedReceiver(msg))
Checksum(pkt)
FieldContainedIn(field,pkt)

Figure 22: List (e) for Levels up to and Including Level 2, New Vocabulary to be Defined Later

Figures 14-17 represent the results of the OHA so far. Further development will modify these four tables. The output of the Preliminary Hazard Analysis performed according to OHA is then the starting vocabulary along with these four tables (as modified through the entire analysis).

Conclusion

We have performed part of a Preliminary Hazard Analysis of a generic communication bus for a road transport vehicle, using the method Ontological Hazard Analysis, OHA. PHA is still a process in which, to put it bluntly, one is asked to sit down and think of as many hazards that can befall a system in operation as possible. The goal is to think of and write down as many as one can. OHA uses formal or semi-formal refinement and strict control of vocabulary in order to control the PHA.

We have used one application of HAZOP, in order to start the HazAn at Level 0. The HAZOP principle of groupthink was not applied; although checking through colleagues is necessary as always, we have found that the major benefits of OHA come from the control that semi-formal refinement brings, and not through subtleties in application of HAZOP guidewords (indeed, it is dependent on the example whether HAZOP be used, and bring benefits, in an OHA).

Most of the results of the HazAn to this point have resulted through the effects of refinement and its control.

The HazAn we have started here using OHA, and not yet concluded, show that a Preliminary HazAn, at even a high level of abstraction such as this, can be sophisticated, requiring sophisticated control, in this case through formal refinement, and an audit trail.

References

[BedCoo01] Tim Bedford and Roger Cooke, Probabilistic Risk Analysis: Foundationa and Methods, Cambridge University Press, 2001.

[Dr03] Kevin Driscoll, Brendan Hall, Håkan Sivencrona 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 on-line at

http://www.cs.indiana.edu/classes/p545-sjoh/read/Driscoll-Hall-Sivencrona-Xumsteg-03.pdf
or through www.springerlink.com for those who are registered with SpringerLink.

[FTH81] W. E. Vesely, F. F. Goldberg, N. H. Roberts and D. F. Haasl, Fault Tree Handbook, Document NUREG-0492, U.S. Nuclear Regulatory Commission, 1982. Available on-line from http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0492/

[Hen97] Dirk Henkel, Safely Sliding Windows, Diplom Thesis, RVS Group, Faculty of Technology, University of Bielefeld, 1997. Available on-line through http://www.rvs.uni-bielefeld.de/publications/abstracts.html#SafelySW

[KamHas99] Daniel M. Kammen and David M. Hassenzahl, Should We Risk It? Princeton University Press, 1999.

[KumHen96] Hiromitsu Kamamoto and Ernest J. Henley, Probabilistic Risk Assessment and Management for Scientists and Engineers, 2nd Edition, IEEE Press, 1996.

[Lad96] Peter Bernard Ladkin, Formal But Lively Buffers in TLA+, Research Report RVS-RR-96-07, RVS Group, Faculty of Technology, University of Bielefeld, 1996. Available on-line through http://www.rvs.uni-bielefeld.de/publications/abstracts.html#Bufferspec

[Lad01] Peter Bernard Ladkin, Foundations of System Analysis, Chapter 3 of Causal System Analysis, 2001. Available on-line through http://www.rvs.uni-bielefeld.de/publications/books/ComputerSafetyBook/index.html

[Lad08] Peter Bernard Ladkin, An Overview of IEC 61508 on E/E/PE Functional Safety, TechnicalReport, Causalis Limited, 2008. Available on-line at http://www.causalis.com/IEC61508FunctionalSafety.pdf

[LadLam99] Peter B. Ladkin, Leslie Lamport, Bryan Olivier, and Denis Roegel, Lazy Caching in TLA, Distributed Computing 12:151-174, 1999. Available on-line through http://www.rvs.uni-bielefeld.de/publications/abstracts.html#Lazy

[LiCo04] S. Lin and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications, 2nd edition. Prentice Hall: Englewood Cliffs, NJ, 2004.

[LitStr93] Bev Littlewood and Lorenzo Strigini, Validation of ultrahigh dependability for software-based systems, Communications of the ACM 36(11):69-80, November 1993.

[Par72] David L. Parnas, On the Criteria to be Used in Decomposing Systems into Modules, Communications of the ACM 15(12):1053-1058, December 1972. Available on-line at http://sunnyday.mit.edu/16.355/parnas-criteria.html

[ReChCa99] Felix Redmill, Morris Chudleigh and James Catmur, System Safety: HAZOP and Software HAZOP, John Wiley and Sons, 1999.

[Sie10] Bernd Manfred Sieker, Systemanforderungsanalyse von Bahnbetriebsverfahren mit Hilfe der Ontological Hazard Analysis am Beispiel des Zugleitbetriebs nach FV-NE, doctoral dissertation, Faculty of Technology, University of Bielefeld, March 2010.

[StSiLa09] Jörn Stuphorn, Bernd Sieker, and Peter Ladkin, Dependable Risk Analysis for Systems with E/E/PE Components: Two Case Studies. In Chris Dale and Tom Anderson, eds., Safety-Critical Systems: Problems, Process and Practice, the Proceedings of the Seventeenth Safety-Critical Systems Symposium, Brighton, U.K., 3-5 February, 2009. Springer-Verlag London, 2009.

[Stu05] Jörn Stuphorn, Iterative Decomposition of a Communications-Bus System Using Ontological Analysis, Diplom Thesis, University of Bielefeld Faculty of Technology, June 2005. Available on-line at http://www.rvs.uni-bielefeld.de/publications/Diplom/stuphorn.pdf

[Tan03] Andrew S. Tanenbaum, Computer Networks, 4th Edition. Addison-Wesley/Pearson Education International, 2003.