Representation of function

Jon Bell

Doc. ref. SD/TR/FR/14: July 29, 2004

Abstract

For automated design analyses (specifically FMEA and SCA) it is useful to have an automated design analysis tool interpret the results of the simulation of the system under analysis in terms of the purpose of the system and the design analysis task. The description of system function (in "functional labelling") has been shown to be a useful approach to this for systems of simple functionality.

This report attempts a description of a suitable representation of function to allow the approach to be used for various design analysis tasks and as the underpinning of a functional description language of sufficient expressiveness to allow systems whose functionality depends on complex behaviours to be described.

1 Introduction

This report sets out the requirements and properties of an approach to representation of function to allow it to be used for interpretation of simulation of engineered systems. As such it forms a sort of preparatory draft of material for the fifth chapter of my PhD thesis. The representation will form the underpinning for a more expressive language than that used by AutoSteve, to allow description of systems whose functionality includes behaviours with the following characteristics.

- Hierarchical function. This is already used in AutoSteve, of course, but a more fully realised hierarchy of function is to be proposed in (Bell 2004b).
- Intermittent and sequential behaviours. These are discussed in (Bell and Snooke 2004).
- Temporal constraints, such as need for function to be achieved within a deadline. This is discussed in (Bell 2004c).
- Functions that depend on the state of some other function, such as telltale functions. This area is covered in (Bell 2004a).

These reports all form early attempts at describing the material in chapters in the thesis, so should be read together. I shall try to avoid repetitive introductory material in each of the other reports, they will refer back to this one.

2 The aim of representation of function

The principal aim of the functional description language to be proposed here and in the companion reports is to allow an automated design analysis tool to interpret the results of simulation of an engineered system in terms of its purpose. It should allow the significant aspects of the system's behaviour to be identified. It is hoped that this representation will also be useful for other related tasks, such as helping with diagnosis, but that is not the aim of the present work. It should therefore be appreciated that what matters here is that the language devised can be used for this interpretation rather than being an ideal representation of function. As different researchers regard the idea of function differently, reaching a consensual ideal representation of function would be impossible. (There will, of course, be more on this material in earlier chapters of the thesis.)

The aim of this can be broken down into these requirements.

- An abstract representation of behaviour sufficient to show when a purpose of the system is or is not being achieved.
- Some representation of the purpose, such that the consequences of the failure of a function can be identified.
- Representation to be applicable to design analysis tasks, both failure analysis (e.g. FMEA) and design verification (e.g. SCA).
- Representation capable of decomposing function so that the relationships between several inputs and outputs can be described.
- Representation to be able to support the description of system functions that depend on intermittent and sequential behaviour.
- Representation to be able to be extended to describe untimely (e.g. late) achievement of a function.
- Representation to be able to describe functions that depend on some other function, such as telltale and backup functions.

This piece considers the earlier points in the above list. Each of the four last points is discussed separately. However, clearly the framework for functional representation must allow for these extensions.

3 Characteristics of function

Function, according to (Chittaro and Kumar 1998), is an overloaded concept. The 'purposive' definition defines function as a relation between behaviour and purpose while the 'operational' definition defines it as a relation between input and output. Here we are more concerned with the former but not to the exclusion of the operational definition.

A useful (if crude) definition of function might be "The behaviour of a device expressed in terms of its purpose". This can be regarded as leading to specific features of system function. As the purpose of a device will be separate from the device (external to it) this seems to imply an external view of behaviour, suggesting that behaviour is represented as a relationship between the inputs and outputs. This gives us two layers to a view of function: the behaviour (seen as a relation between input and output) related to the purpose (by the definition). This could be argued to combine the two uses of the term in (Chittaro and Kumar 1998). The idea of an external view of the behaviour is not inconsistent with the idea of function as effect. The idea that we can represent a system's behaviour in terms of whether or not it is fulfilling its purpose is a useful abstraction for generation of design analysis reports (such as for FMEA). The usefulness of this abstraction for subsystem function seems to me to depend on the design analysis task (it might be useful for design verification but less so for failure analysis). How does this relate to its usefulness for diagnosis? There is room for more discussion here.

It seems worth trying to fit this definition with appropriate definitions for the other classes of knowledge in (Chittaro and Kumar 1998). I propose the following:-

Structure The physical composition of the device; its components and connections.

Behaviour How a device works, what it does in terms of its internal properties.

Function A device's behaviour expressed in terms of its purpose.

Purpose The need that the device is intended to fulfil.

With the suggested split between simulation and interpretation, there is a definite distinction between behaviour and function, behaviour being on the simulation side and function on the interpretation side. This might be affected if a functional model resulting from simulation of some system was used in simulation of a larger system of which the earlier system is a part, but even in this case, it is preferable, at least for failure analysis, to use a behavioural rather than functional model of the subsystem. This, of course, because for the simulation we are interested in what actually happens, the abstraction of behaviour in terms of purpose is no longer appropriate.

One possibly interesting approach to distinguishing between the four classes of knowledge is:-

Structure What is inside the device.

Behaviour What happens inside the device.

Function What happens at the surface of the device (or the boundary of the simulated world).

Purpose What goal is enabled outside the device.

This does draw a reasonably clear distinction between behaviour, function and purpose which seems not inconsistent with other approaches, though perhaps the idea of function as effect in (Chandrasekaran and Josephson 1996) is veering towards the outside viewpoint. I do not suggest these are complete as definitions, there is no idea of intention in these, though it is implicit in the idea of purpose. These viewpoints moving out from the inside to the surface to the outside of a device are certainly consistent with the definitions above. It is also consistent with the idea that each class of knowledge abstracts the previous. I remain uneasy with this notion because for failure analysis, at least, we need to consider alternative behaviours, which even if we map to unintended functions can hardly be sensibly mapped to unintended purposes. I suggest that we cannot simply replace the alternative behaviours with the idea that the function (or purpose) has failed, as we will want to know what did happen as well as what didn't but should have.

The purpose of a system has a clear connotation of its relationship with its environment. This fits in well with the idea of expressing a system's function in terms of inputs and outputs, an external view of behaviour.

The idea that function expresses the purpose of a system implies some notion of intent, the idea is to capture what the system is designed to do. This in turn suggests that unintended behaviours are not necessary to the functional model. There are difficulties here, concerning unintended outputs of a system, either as the result of a malfunction or as a side effect of its correct behaviour. However, if we can limit functional description to capturing intended behaviour this does simplify the functional model, reducing the overhead of user input.

The idea that a system function is concerned with inputs and outputs to the system and to its purpose (intended behaviour) leads to the idea that the input is (generally) going to be of a form that expresses the environment's (that is, typically, the user's) intention for the system, so the function is not triggered by the input of energy (say) but rather the input to the system's controls, so the preconditions for the achievement of a system function are likely to be expressed in terms of switch positions. To illustrate this idea, consider the effect of modelling the input of energy to an electrical system. There are broadly two cases — either the system is self contained, having its own power supply (a battery) in which case there is no energy input or the power supply is external and can (generally) be taken for granted. Do designers of mains electrical appliances need to model the reliability of the public supply? It seems rather outside their scope. There is an exception, of course, where failure of the power supply leads to some alternative system behaviour (using a backup supply), such as a battery powered emergency lighting system coming into effect on failure of mains power. In this case failure of the mains supply must clearly be modelled or the backup system will not be analysed.

This idea of input and intention seems to lead to a clear distinction between system level function and component level function. A system function can be defined in terms of controlling inputs (defining intention) while a component level function is to depend on inputs of energy or whatever. A lamp's function is more or less "if you put electricity in, you get light out". Arguably this is closer to an abstract view of behaviour — it describes what a lamp does with no specific relation to purpose, which is broadly a system level idea. Why a lamp should be required to light will depend on its context, which in turn depends on what the system is for, what its purpose is. The idea that a function relates to a purpose means that a system might well have several functions that will not be conveniently separated into (structurally) distinct subsystems. For example a car's exterior lighting system has the purposes (and therefore the functions) "light road ahead", "make car visible", "show braking", "show intended change of direction" and others. All the listed functions have some components in common with others. The lamp switch is used in all but "show braking", and that function might share the tail lamps with the "make visible" function, as is the case in our car.

It seems worth briefly discussing what are adopted as characteristics of function for the purpose of the descriptive language being devised. A working definition might be "the behaviour of a system expressed in terms of the purpose of the system". The properties are taken to be inputs to and outputs of the system. The system, in turn, is taken to be that which is to be simulated, so as to gain a description of its behaviour. As a system will have a purpose external to itself, it is intended to cause an effect on its environment, the idea of functional description is to provide a description of the interface between the system and its environment. Therefore we are concerned with those parts of the system that have a direct effect on, or are directly affected by, the system's environment. The idea of using the notion of function to interpret the behaviour of a system vis a vis its environment implies that the system's behaviour is to be derived from the simulation, but the environment's will not be so derived. In other words, this use of function can be regarded as a means of representing knowledge that is not possessed by the simulator. This knowledge might conceivably be from many fields, besides the physical, such as of the legal consequences of certain system failures.

If function is to be used as the interface between the simulated system and the unsimulated environment it seems useful to picture it at the edge of the simulated world, as distinct from the behaviour and structure, which are internal to that world and the purpose which is external.

At the risk of stating the obvious, hierarchical function is essentially concerned with system functions that depend on more than one of either input settings or output effects or both. If the function has but one input setting and one output, there is no gain in attempting to decompose it.

Another aspect of function is that it is frequently independent of of the system simulator's domain. For example, the functions of gas and electric cookers are identical. If function is considered at the boundary of the simulated world, then the simulation's domain is clearly irrelevant. There might well be cases where the function will be within the domain, such as an electrical back up power supply.

4 Logic and system function

If we are to use function to identify when a system is (or is not) achieving its purpose, we need some way of recognising how the system should achieve its purpose. The approach used by AutoSteve is to use a form of logic in which achievement of a function is associated with its effects, in other words the outputs (or goal states) of the system. This was used for FMEA, with the addition of noting the consequences of failure of the function. For SCA, the expected inputs (switch positions) had to be added as these could not be derived from the correct (that is with no failures) simulation of the system as they can for failure analysis (such as FMEA).

This section will attempt to put the logic that underlies recognition of achievement of function on firmer ground. It will illustrate the discussion using a very simple system, a torch. The torch has, essentially, two interfaces with the outside world, the switch, which can be either on (closed or of negligible resistance) or off (open, of infinite resistance) and the lamp itself which is lit if there is (sufficient) electric current flowing through it. The torch has but one function, which might informally be stated as "when the switch is 'on', the lamp will be lit". There are four external system states - switch off or on and lamp lit or not. Only one of these results in achievement of the function. We can regard the switch position as the precondition for the function's correct achievement (its trigger) and the lamp being on as its postcondition (effect). [1] The function is achieved if both pre- and post-conditions are true. However, AND is not sufficient as we need to distinguish between the cases where the function is not achieved. This is illustrated in Table 1. In the case of the torch, if the switch is off and the lamp not lit, the

Precondition	**Postcondition**	**Function**
false	false	not intended
true	false	failed
false	true	unexpected
true	true	achieved

Table 1: The four truth states of a function

function is not wanted, if the switch is on and the lamp lit the function is achieved; these are correct behaviours, consistent with the intended functionality. However, if the switch is on and the lamp not lit (perhaps the battery is flat) then the function has failed while if the switch is off and the lamp lit then the function is achieved unexpectedly. There seem to me to be objections to the phrase "function achieved unexpectedly" which will be discussed later. It will be seen

[1] I propose to use 'trigger' to mean precondition (in the sense of input) and 'effect' to mean post-condition (or output or goal state) in these reports. This avoids the need to distinguish between the alternative terms or senses. I hope that the sense make the meaning clear. I avoid the term 'cause' as this model of function has no notion of causality, this being derived from the simulation.

that where both pre- and post-conditions are of the same value, the behaviour is consistent with correct achievement of the function. A system can fail in two ways: either the expected outputs do not occur or (less commonly) the outputs occur when not expected. We want to distinguish between the two other cases as the consequences are different. In the case of the torch, if the function fails the user cannot see but if the function is achieved unexpectedly the battery will be drained.

This suggests a need for there to be four possible values for the achievement of a function, as shown in the table. There seems to be a need for a term to separate the pre- and post-conditions of the function, TRIGGERS is used herein. This is only needed for writing textual statements as a representation of function might, internally, simply list the preconditions separately from the post-conditions. All functions will (implicitly) have pre- and post-conditions and the relationship between them does not alter. In any function, if the precondition (trigger) is true and the post-condition (effect) false than the function has failed. The pre- and post-conditions can be boolean expressions, which allow several input and output conditions to be combined. For example, some torches have a slider switch for if the torch is to stay on and a push button so that it can be flashed (for signalling, perhaps) in which case the function might be informally stated as "if the slider is on or the push button is pressed the lamp is lit". The decomposition of function is discussed in the companion report (Bell 2004b).

It should be made clear that this is hardly news, this logic is used in AutoSteve. This is really only an attempt to provide a basis for its use, perhaps to define it a bit more formally than has been done.

In the torch example above, the consequences of the two possible failures were suggested. We can suggest that the purpose of a torch is to be a source of light, in which case the consequence of the failure of the function (no light) can be expressed in terms of the purpose of the torch, so is consistent with the idea of function as a mapping between behaviour and purpose. However, the consequence of the function being achieved unexpectedly (drain on the battery) cannot be expressed in terms of the purpose of the torch. This is the problem with the idea that the function is achieved unexpectedly. Indeed, if we are to consider function as a relation between input and output, then the function is not achieved as the input is absent. This is, of course, consistent with the idea that the post-condition being true when the precondition is false is a faulty behaviour. This illustrates the idea that what is achieved unexpectedly is the effect (output or goal state) associated with the function. This distinction might seem rather pedantic but it leads to a change in how unexpected achievement should be described - it should be associated with the effect of a function, not with the function itself. This will be discussed in more detail in (Bell 2004b), but it is worth noting that where a function depends on two outputs (such as both headlamps being lit) then if one output stays on continuously, the consequences of this behaviour are missed if they are associated with the function as the function is not achieved when only one is found, so is not achieved unexpectedly when only one output is present.

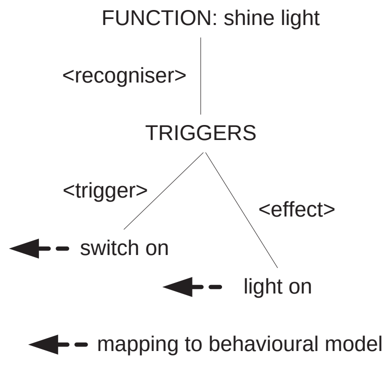
To represent a function as a mapping between behaviour and purpose such that the two types of failure can be distinguished we need to represent the behaviour in terms of the triggers and effects and the purpose. The purpose might be explicitly represented, but in AutoSteve is represented by consequences associated with failure of the function or its unexpected achievement. The representation of purpose will be discussed in the following section.

5 Representation of function

If function is to be regarded as a mapping between behaviour of a system and its purpose and also between the inputs and outputs of the system this means we need elements in the representation that reflect these mappings. I propose to use the term attribute to mean an individual element of a functional description. The term component might lead to confusion between the idea of a component of a function and the physical component of the system, using attribute should avoid this. Chris (Price 2004) seeks to represent function as a triplet of name, recogniser, export. The name can, of course, provide an informal mapping to purpose by suggesting that purpose (for example "light road ahead") but something more specific is needed for design analysis, though perhaps not for diagnosis — by then you know what the function is for! However, an identifier is (presumably) required. An interesting point that arises from the use of the function's name (identifier) as an informal mapping to purpose is the idea that this highlights the idea that one use of the idea of 'function' is to map between the informally stated idea of purpose and the formal derivation of the system's behaviour. There seems to be a rough parallel here with formal methods of specification the need for which is arguably an aspect of the need to establish a formal definition of an initially informally stated requirement. Does this suggest that a functional language might be used in the requirements definition stage of the design process to formalise the description of the required behaviour of the system?

At the level of functional representation considered here, there seems to be no need for the 'export' property a boolean to show whether or not a function is exported. For the interpretive purpose of function, in which function is taken as mapping (an external view of) the behaviour of the system to its purpose, then arguably the export of the function is inevitable. The hierarchy of function is more extensive in (Price 2004) than is required for interpretation of a system's behaviour, this difference seems to account for the addition of the export attribute.

The approach to logical representation of achievement of function introduced in section 4 above provides a basis for the recogniser, whose rôle is to establish the state of achievement of the function. The recogniser is therefore divided into two parts, the trigger and effect, the pre- and post-conditions viewed from the logical perspective and inputs and outputs from the system point of view. However, for failure analysis the trigger can be derived from the simulation of the correct system. The function and recogniser for the torch are illustrated in Figure 1. The



POSSIBLE TEXTUAL FORM:

FUNCTION: "shine light"
FAILURE: no light, user not helped to see
RECOGNISER: "switch on" TRIGGERS "light on"

Figure 1: The function of a simple torch

triggers and effects provide the 'hooks' that allow the functional description to be linked to the system's behaviour being identified with the inputs to, outputs from or states of chosen components. This is as done by AutoSteve, of course, although the representation of triggers in AutoSteve is confined to this link, while it is suggested that the triggers form part of the functional model itself (necessary for design verification). Another aspect of the linking between the functional model's effects and the behavioural model's outputs (or goal states) is the idea that the link should be one to one, so that the functional description proper is complete and so that individual outputs cannot be missed. There is more on this in the report on functional decomposition.

How best to represent the mapping to purpose is an interesting question. AutoSteve, of course, does this by including the consequences both of failure and unexpected achievement of a function and this might well remain the best approach. An alternative would be to list the consequences of achievement of a function, both good and bad, and to have the good consequences shown as lost for failure of the function and the bad as present for unexpected achievement of the function. I am uncertain of there being any advantage to this approach and it looks less intuitive than using consequences of failure.

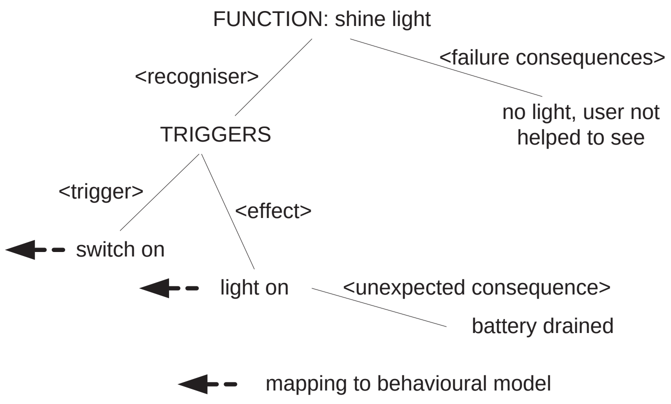
The use of consequences has the advantage of reducing the asymmetry between the failure and unexpected achievement of functions (or outputs). This is because the consequences of unexpected output cannot be expressed in terms of the achievement of a purpose of the system, broadly that would amount to the function being unnecessarily achieved which isn't very interesting. The idea of attempting to list all the aims and consequences of a function also tends away from the idea of attaching the consequences of unexpected achievement to effects (outputs) as suggested above and is helpful for decomposition of function.

One aspect of representing purpose in terms of consequences of failure is that there might be several possible failures where the output (effect) is not binary, or a set of binary outputs. For example, an oven might have a function "cook casserole" which is achieved if oven control is set to low and, crucially, the temperature remains low. There are two possible failures, of course. Either the oven fails to heat up (sufficiently) or a fault in the control system (gas regulator or thermostat) allows the oven to run hot. It might even be required to use three — no heat, too cool, too hot, depending on how detailed the modelling is to be (and what failures there are). The consequences are clearly different in these cases. There is room for more thought as to how this is managed, though I imagine that the idea of mapping ranges of acceptable output (in this case temperature) to achievement of function or specific consequences (much as Dougal did) is the way to go here. One interesting point that occurs here is that these alternative failures can be modelled qualitatively for failure analysis (the correct temperature being used as a landmark value) but less readily for design verification. This example is complicated by the fact that temperature might be a dynamic, abstract state in the manner of (Keuneke and Allemang 1988).

Incidentally, this example is a good illustration of the domain independent nature of function description. It can be used for either a gas or an electric oven, the only difference being that the control setting low will be mapped to, say 150 degrees for an electric oven or gas mark 2 if we are cooking by gas.

While it certainly seems not impossible to have the mapping to purpose represented by a description of purpose (and side effects) of function and using this to derive the consequences of its failure, there seem to be reasons for using the AutoSteve approach of representing this relationship by describing the consequences of failure of a function or unexpected achievement of one or more of its effects. This modelling of function for the torch is shown in Figure 2.

One problem with the association of unexpected consequences with effects in the functional model, as above, is that it weakens the domain free nature of the model. This is because the



POSSIBLE TEXTUAL FORM:

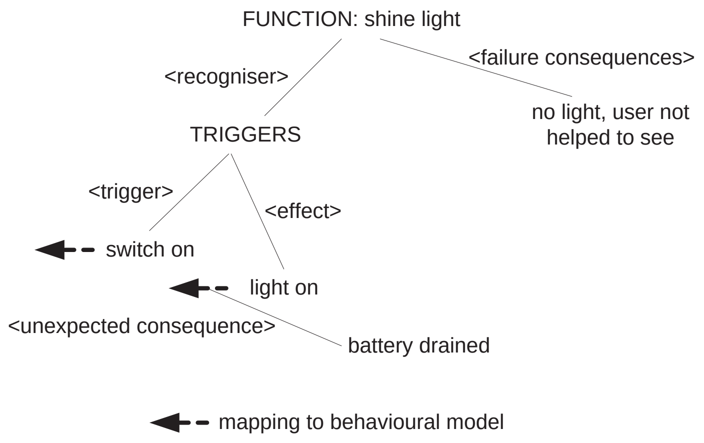
```
FUNCTION: "shine light"
   FAILURE: no light, user not helped to see
   RECOGNISER: "switch on" TRIGGERS "light on"

EFFECT: light on
   UNEXPECTED: battery drained
```

Figure 2: The torch function with consequences

unintended consequences are likely to be related to the domain of the system, such as the draining of the torch battery. One possible way of mitigating this might be to attach the unexpected consequences to the mapping between the functional model and the system behaviour as shown in Figure 3. The arguments between this approach and attaching the consequences to the effect are largely pragmatic; which better suits reuse of the functional model? There is no reason why a functional model should not reusable, the mappings between function and behaviour being specified for each new use of the model. If the unexpected consequences are attached to these mapping then they need re-stating for each reuse of the model. However, if they are attached to the functional model, it is no longer domain free. In practice it seems arguably to use the latter approach as the cases where the same functional model can really be used across different domains seem to be relatively few and far between.

While a full representation has three attributes — identifier, recogniser and mapping to purpose, with the recogniser subdivided into pre- and post-conditions, the need for these is dictated partly by the design analysis task. For failure analysis there is (generally[2]) no need for the precondition as this can be derived from the simulation of the system behaving correctly. Indeed, where this is possible basing the comparison between correct and failure mode behaviours is more properly done by comparing simulations, as here the concern is purely with what the system does. The effects of failures in the system should be compared with what does happen in the correctly working system, not what should be happening. Using the specified inputs could be argued to blur the distinction between failure analysis and design verification. How the presence of systems whose correct behaviour cannot be unambiguously determined from a 'correct' simulation will be considered in the report of functional dependencies, (Bell 2004a). For diagnosis (where the functional failure is indicated by a symptom of an unexpected (lack

[2]Where a function depends on more than inputs to the system, this is not the case. More in report on functional dependencies



POSSIBLE TEXTUAL FORM:

```
FUNCTION: "shine light"
   FAILURE: no light, user not helped to see
   RECOGNISER: "switch on" TRIGGERS "light on"
```

Figure 3: Attaching consequences to the mapping between function and behaviour

of) output) there is no need for the mapping to purpose. By then we know what the system is for! Unfortunately from the labour-saving point of view this does not help much. The idea that we can rely on the simulation of the correct behaviour to derive the preconditions for functions assumes that the system is indeed correct (implying the design is verified). A reasonable model of the design process has design verification preceding failure analysis preceding use of model for diagnosis and all parts of the functional representation are needed for design verification.

6 Subsystem function

While the model of function proposed, in which function forms the interface between the (internal) structure and behaviour of the system and its (external) purpose, implies that functional representation is at the boundary of the system, there might be cases where functional representation might be found at an internal level, once a subsystem is incorporated in a larger system. This is a distinct case — I am not referring to cases where a larger system simply incorporates the functionality of a system as a subset of its own functionality (such as a car's lighting functions being those of the lighting system) but rather cases where the functional boundary of a previously analysed system is now internal to a larger system. One possible case is where an electrical system is divided into "control" and "power" sides, each designed by different groups. In this case the control system might have as its effector components various relays that carry out the switching functions for the power system, so the goal states of the control side are the positions of the relay switches. Once the design of a system has been verified, it seems quite possible to use its functional model as an abstract behavioural model for simulation of a larger system in which it is incorporated, but with certain caveats. These include:-

- The subsystem is to be simulated as working correctly. The functional description does not describe what happens if there is a failure within the subsystem.

- There are no outputs (effects) of the subsystem that are not included in its functional model that will affect the simulation of the system as a whole. Examples might be side effects of the operation of the subsystem. How much of a problem this is seems likely to depend on how sophisticated the system simulation is.

It seems entirely possible that the abstract view of behaviour as used in this functional representation could be generated from simulation of the system so as to generate similar abstract behavioural models of a system under failure modes. These might be used in simulation of a larger system. However, it will still be the case that unexpected outputs (i. e. side effects) will need to be included. I imagine such a model being generated from an attainable envisionment of the system rather as is proposed for design verification, and the external view of the resulting behaviours be abstracted from that using the functional model of the system as a starting point in identifying which component states should be included. This might also be an approach to generating a fault tree of the system.

6.1 Side effects and function

If function is concerned with purpose, this leads to difficulty modelling function based on an output that does not relate to the outputting component's purpose (a side effect). One obvious example is a car heater relying on a heat exchanger to use hot engine coolant as the source of heat. According to the definition used here, it is incorrect to suggest that this heat generation is a function of the car engine — it is not its purpose. It seems to me rather to be be aspect of the behaviour. However, it does fit with the idea of function being partly an external view of behaviour, the heat is certainly an output and so could affect the engine's environment. These distinctions might in certain circumstances break down. On a cold night the driver of a car held up by a blocked road might well run the engine simply as a source of heat to keep the passengers warm (if there is plenty of fuel available!).

If one is modelling the heating system, it seems not impossible that the engine is regarded as an independent source of heat (the designer of the heater is unlikely to be concerned that the heat depends on, say, the availability of fuel). On the other hand the design analysis of the whole car will include the engine and the heat output can be obtained from the behavioural model of the engine — it is internal to the behavioural (simulated) world. I suggest, therefore, that if we regard the functional view as being of the boundary between the system and its environment, the problem of side effects is more or less finessed. This might be different where functional decomposition is used independently of behavioural simulation in diagnosis, however. Even here though I wonder how important this is. The absence of a side effect is unlikely to be the most obvious symptom, though the correct working of the source of the side effect is presumably an easy way of eliminating possible causes of the failure. To use the same example, diagnosis is hardly going to start from the observation "the engine can't be working because the saloon never gets warm" simply because there are other more important effects of the engine not working. Even if the cooling system has failed (a radiator leak) than the engine will display its own symptoms.

The proposed functional representation does not rule out the inclusion of side effect outputs. They are arguably out of place in the recogniser, strictly speaking, but as they are (presumably) inevitable their presence does not affect the truth of the recogniser, at least for normal operation. As an internal combustion engine gets hot whenever it is running, the addition of heat to the outputs associated with the functional model means that the recogniser is still true whenever the engine is running. This allows the addition of side effects to the functional model (e.g. for functional decomposition for diagnosis) without affecting the model's use for interpretation of behaviour if they are required for other uses of the functional model.

This is, of course, a different case from a malfunction leading to some unexpected output (maybe an electric motor overheating when stalled). As the functional model is concerned with whether or not the purpose is being fulfilled and not with descriptions of other behaviours, this case will not be covered by the functional representation and must be handled by the behavioural model. It is, I suggest quite consistent to model this case as an aspect of component (or system) behaviour.

7 Conclusion

The representation of function proposed here is not very different from that used by AutoSteve. The main difference is the moving of the consequences of unexpected achievement away from function to an effect. This has the advantage of ensuring (as far as possible) that all unexpected outputs are noted in FMEA, avoiding the case where an output that does not itself constitute a function is missed. I am aware that I have not covered this area fully, it really belongs in the functional hierarchy report.

There is an arguable difference between this proposed approach and AutoSteve on the inclusion of pre-conditions. AutoSteve seems to assume you needn't bother because it starts from FMEA, while I assume that you should, but might get away without for certain analyses. I can hardly claim this is important, though I wonder if this doesn't lead to a minor difference in that here preconditions or triggers are an intrinsic part of the functional model (even if derived from simulation) while in AutoSteve they seem to be more a property of the mapping between a functional model and a specific system. This might affect the reusability of functional models between systems while improving its usefulness for different design analyses.

The approach suggested avoids the use of 'incorrect' functional models. They are unnecessary for the interpretive rôle (as the use in AutoSteve suggests) and so simply add complication to the construction of functional models. This is, of course, similar to AutoSteve.

There is no problem adding further attributes to the functional representation proposed. Two obvious examples are the 'export' attribute from (Price 2004) and inevitable side effect outputs that can be added to the effects of the recogniser without affecting its use as they will always be true.

I do not claim that there is much new here, it really is intended to form the basis for the new material in the other reports (chapters). I hope it does this.

References

Bell, J. (2004a). Dependencies between functions. SoftFMEA doc. ref SD/TR/FR/09.

Bell, J. (2004b). Hierarchy and function. SoftFMEA doc. ref SD/TR/FR/12.

Bell, J. (2004c). Time and function. SoftFMEA doc. ref SD/TR/FR/13.

Bell, J. and N. A. Snooke (2004). Describing system functions that depend on intermittent and sequential behavior. In *Proceedings 18th International Workshop on Qualitative Reasoning, QR2004*.

Chandrasekaran, B. and J. R. Josephson (1996). Representing function as effect: Assigning functions to objects in context and out. In *Proceedings of American Association for Artificial Intelligence.*

Chittaro, L. and A. N. Kumar (1998). Reasoning about function and its applications to engineering. *Artificial Intelligence in Engineering 12*(4), 331.

Keuneke, A. M. and D. Allemang (1988). Understanding devices: Representing dynamic states. Technical Report 88-AK-DYNASTATES, Ohio State University. From the Laboratory for Artificial Intelligence Research, Department of Computer Science.

Price, C. J. (2004). Abstract modeling for vehicle diagnosis. Still to be submitted for publication.