Towards a language for interpretation of simulation of engineered systems

Jon Bell

Doc. ref. SD/TR/FR/11: April 22, 2004

Abstract

If an automated design analysis tool is to generate a useful (that is, human readable) report describing the results of its simulation of the subject system, the simulation needs interpreting in terms appropriate to the purpose of the system and to the design analysis task being undertaken. Previous work in Aberystwyth has made use of a functional modelling language for this task, which language has been incorporated into a commercial design analysis tool. This language does have shortcomings, however, notably its inability to capture temporal aspects of the behaviour of the system, either intermittent or sequential behaviours or late achievement of a required system function, and possible difficulties arising from cases where a system function depends on the achievement (or failure) of some other function.

This report discusses extensions to the existing language that allow these problem functions to be modelled, borrowing operators from temporal logic. It also suggests some minor alterations to the existing hierarchical functional modelling, the better to capture relationships between functions.

1 Introduction

This report is intended to focus a more or less "top down" approach to devising and describing a possible version of a functional modelling language that is more fully realised than that used by AutoSteve. It is not the intention to discuss any specific area in great detail, but more to allow the relationships between different aspects of the proposed language and approach to be made clear. It is intended to focus discussion, as such I have not avoided some possibly controversial (read daft) ideas.

In the interests of brevity, I do not propose to discuss definitions of function at any length, that stuff can wait until a future report, or thesis. However, I shall briefly introduce a possible (and hopefully useful) definition in the next section, on requirements of the language. Instead this effort will concentrate on actual aspects of the extended language. There will be a sections addressing the three areas where some modification of the existing language might be required. These are sequential behaviours, timing and dependencies between functions. These follow a section discussing the hierarchy of function. This leads on to the pragmatic point that what is important about this approach and the language involved is not how well it represents function but how capable it is of being used for interpretation of some appropriate simulation.

2 Requirements

The first thing to note here is that what is required is a language to support the interpretation of system behaviour in terms of its purpose, and the design analysis task being undertaken.

Whether such a language is to be regarded as a full representation of the system's function is (arguably) a subsidiary question. However, this mapping between behaviour and purpose is consistent with the idea of the "purposive" definition of function in (Chittaro and Kumar 1998) and so I propose to continue to use the term "functional language". The requirements of such a language are similar to those in (Pugh, Price, and Snooke 1995), simplicity and reusability. Naturally if the proposed language is to be implemented as an extension to the existing language, it should also preserve compatibility withe the existing language, so existing functional models can be reused. This requirement helps focus on the simplicity requirement, of course, as the extensions are concerned with modelling more complex systems so back compatibility ensures that simple systems' modelling becomes no more (needlessly) complex.

The proposed working definition of function that this language is intended to use is "The intended purpose of a system, expressed in terms of system properties". This definition is not, perhaps ideal, so it seems worth listing characteristics of function, as they seem to correspond with this definition and the intended use. These include:-

- Concerned with externally visible characteristics of the system. As the system's purpose is external to the system (the need it is intended to fulfil) and (component) behaviour and (system) structure are internal, the aim of a functional representation is to bridge the gap. This places functional characteristics at the edge of the system. This idea seems to be (perhaps rather informally) consistent with the use of input and output properties in AutoSteve. What are they if not an attempt to define those parts of the system that interact with the outside world?

- Domain free. The external characteristics of a system might well be in a different domain from the internal workings of the system. A torch's output is light, but it can readily be simulated using an electrical domain simulation tool.

I note that in practice the boundary of the system is likely to be the boundary of the simulated world. Is this inevitable? I suspect not necessarily, but (almost) invariably. There is at least a case to be made for the idea that interpretation of simulation is only necessary because the simulator does not simulate the whole world.

3 Hierarchy of function

The definition of function suggested above in turn suggests that there are three elements to (complete) representation of the function. These are a representation of the function's purpose, of the pre-conditions (input states) that trigger the function and the post-conditions (output states) associated with its achievement. This can, with perhaps a certain amount of stretching of a point, be considered to mean that the representation of function captures both the idea of a function as a mapping between input and output as well as between behaviour and purpose, the two mappings suggested in (Chittaro and Kumar 1998). Both pre- and post-conditions are needed to distinguish between the two possible failures of function, as illustrated in Table 1. This need to distinguish between these failures tends to complicate the hierarchical modelling as it becomes necessary to continue the functional hierarchy down to a (subsidiary) function associated with each individual aspect of the goal behaviour (post-condition), so the headlamps dipped function must consist of left headlamp dipped and right headlamp dipped so that if one headlamp stays on, that is expressed at a functional level, even though (arguably) there is no such function - there is no purpose in ever having one headlamp lit without the other. I realise that here, as in any specific case, this point is arguable, but the principle, I claim, holds.

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

Table 1: Failures of system functions

This approach to hierarchy does differ rather from the approach in (Snooke 1998). That hierarchy seems to include internal (component) function, which this approach does not. That makes the hierarchy in (Snooke 1998) more powerful as a stand alone source of knowledge at the expense of complicating the necessary functional knowledge, and so requiring more input from the user. The approach proposed herein (which is a direct descendant of the approach adopted in AutoSteve) assumes the presence of sufficient knowledge in other classes (behaviour) to use that to generate a representation of system behaviour (the aim of the simulation side of the tool).

The functional hierarchy (or decomposition) is, of course, based on the use of the logical relations AND, OR and XOR, together with NOT, though as a unary operator, NOT perhaps has a less significant role in decomposition. There are a few points worth making about each relation...

OR Might be used to define relations between functions, or within a function's expressions for pre- or post-conditions. It might be the case that there are several functions that can be used as alternative ways of achieving a specific purpose. For example, a boiling a saucepan of water can be done on any ring of a stove, so a failure that results in the loss of use of one ring is less severe than one that means all four fail. It seems possible that a functional model of a stove should capture this distinction, so OR might be used to map each function to the boiling water purpose. Note that the functions are separate, however, and the pre- and post-conditions do differ.

XOR I finally came up with a possible use for XOR, modelling a domestic light switched using two two way switches, so that any change to either switch causes the lamp to change state.

AND An interesting aspect of the use of AND is that it might cover two distinct cases, where a single output (i.e. partial fulfilment of the function's post-condition) is better than nothing and cases where it isn't. Contrast, for example, the Belt Minder warning function, consisting of audible and visible warnings, with the left and right headlamps as components of the headlamps dipped function. In the case of Belt Minder, it seems reasonable to suggest that one without the other is a partial achievement of the warning function, so if the dashboard lamp lights without the chimer sounding, at least the driver gets some warning (unless he is blind!). In the headlamps case, in contrast, it seems arguable that one headlamp being lit does not amount to partial achievement of the headlamps lit function, if only because of the legal implications. I suggests that there are therefore two distinct subclasses of AND, an approach to handling this is outlined below.

This leads me to the idea that there are four distinct elements to a functional hierarchy (or decomposition), at the top is a purpose (which could, as outlined above be achieved by alternative functions). Next comes a function, that achieves the purpose. I think (though am as yet unsure) that a function will be associated with one purpose. A function is (typically) represented by some expression of purpose (in terms of the consequences of it failing when expected) and its preconditions and post-conditions. I appreciate that in most cases the preconditions can be dispensed with for failure analysis, but see the section of functional dependencies below. This representation of function is illustrated in Figure 1. Alternatively to the post-conditions, a

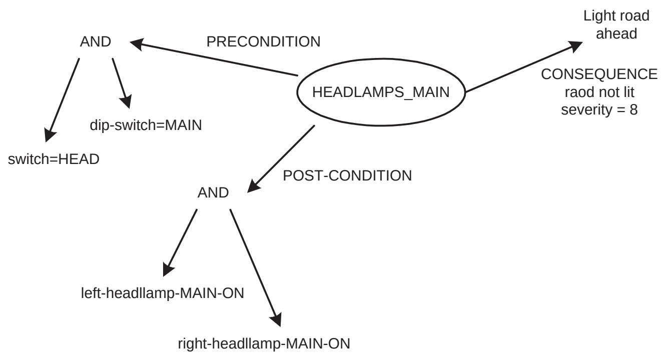

Figure 1: Function maps between purpose and behaviour for pre- and post-conditions

function might be composed of subsidiary functions, in cases where achievement of any of the subsidiary functions is deemed by the model builder to represent partial achievement of the function, so that the severity of the failure is reduced. The belt minder example discussed above is a possible case in point. The pre and post conditions come at the bottom of the hierarchy and will map across to aspects of the system behavioural model in much the way that AutoSteve uses the input and output properties. These are likely to be Boolean expressions, as a function's post-condition will frequently depend on several behaviours, as might its precondition. If a function is a mapping between behaviour and purpose, this achieved by the consequences of failure expressing the mapping between function and purpose and the pre and post-conditions mapping between function and behaviour.

The idea is that subsidiary functions are only used as such in cases where they add information, specifically that a (higher level) function is partly achieved by such a subsidiary function. It will, therefore, have its own consequence of failure and its own value for severity (and possibly also detection). This is illustrated in Figure 2. As a subsidiary function is associated with a higher level function, it has no need of preconditions, it inherits them from the higher level function. It is distinguished from a post condition, however, by having a consequence of failure (and a severity value). Its achievement can, of course, be capable of further decomposition, either as a further subsidiary function (though I suspect this will be vanishingly rare!) or a complex post-condition that maps to more than one aspect of system behaviour. A subsidiary function is therefore distinguished from a post-condition by having its own consequences of failure (and its own severity value).

Another suggested change in the hierarchy is that RPN values for unexpected achievement are associated not with function but with outputs, the arguments being (a) that it is an output that occurs unexpectedly, not a function (which depends on the input) and (b) that this allows subsidiary functions to be dispensed with except when wanted, as discussed above.

One question that needs more thought is whether a function might be composed of other functions - in other words whether there are cases where two or more functions (each with

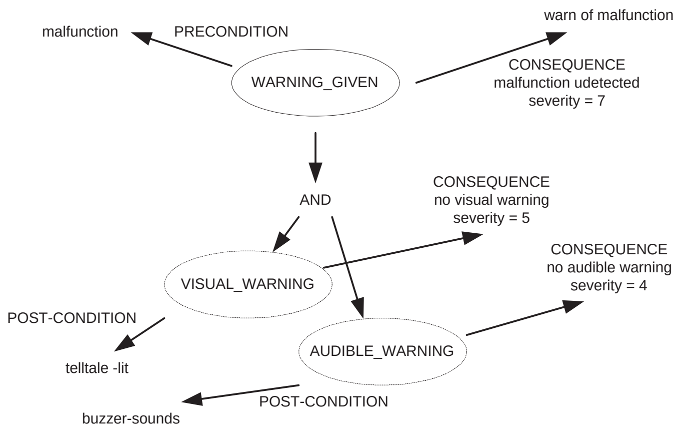

Figure 2: Subsidiary functions allow partial fulfilment of the main function

their own purpose and pre- and post-conditions) might be used together to achieve some other purpose.

4 Intermittent and sequential behaviours

This has been described in the (rejected) aaai paper, since submitted to QR-04, so this section can be kept brief. It does seem worth summarising that work here, partly as background to the next section on timing and function.

Simply, the idea was that the hierarchical functional modelling in AutoSteve is (implicitly) based on a simulation occupying one time slot that starts with the "input event" (that is the function's preconditions becoming true) and ends once the system settles into a steady state. The interpreter then checks whether any function's post-conditions are true and labels the report accordingly. The temporal operators SEQ and L-SEQ were added to enforce the division of this time slot, so each occurrence of these operators marks the beginning of a new time slot. SEQ is used where the succeeding system state must become true in an immediately following time slot (the "next time" operator) while L-SEQ is used where there can be intermediate time slots, wherein the system state is undefined. This means that where AND is used in the hierarchical description following SEQ, all changes must be simultaneous and also that any possible alternative component states cannot occur (so a motor must, for example, switch directly from off to fast). With L-SEQ neither of these are enforced so where a component has several alternative properties, any of them might be true in the intermediate state (so a motor can build up speed gradually). Of course, in principle, a DC motor could even flip briefly between running forwards and backwards in this case.

To avoid this difficulty one idea (which may well not be worth pursuing, it depends how real the problem is) is to have an intermediately constrained version of the sequence operator. One possible approach would be to have the component properties concerned ordered (so a motor might have five; running back fast, back slow, stop, forward slow, forward fast) and only the specified states or those appearing between them in the ordering are allowable in the intermediate time slot, distinguished by some other sequential relation, call it O-SEQ. Suppose that the motor above is stopped O-SEQ forward fast would mean that an intermediate state in which the motor was running forward slowly was acceptable, but it should not jump backwards.

5 Timing and function

This seems actually to be quite simple, as there seems to be but one sensible approach. There are two alternatives:-

Compare simulations Here the timings of the simulation of the system working correctly and the simulation of the system working under a failure are compared. There are two problems with this approach — the need for a "correct" simulation limits it to failure analysis and some sort of temporal constraint is still necessary to distinguish significant changes of timing. This approach therefore offers no advantage over

Set deadlines In this approach the start of each time slot is (optionally) given two temporal constraints, a time before which the system state after the time step must be true (a deadline) and / or a time before which the succeeding system state should not become true.

It is, I suggest, clear that the second of these alternatives is the one to adopt. It is also clearly the case that when the simulation is divided into more than one time slot (using SEQ or L-SEQ) that the timing constraints must be given relative to the start of the previous time slot, not the start of the simulation. This is because if there are tolerances in timing (such as where a flash must be at least 750 milliseconds but no more than 1250) these tolerances accumulate if each timing constraint is relative to the start (triggering event). This is illustrated in Figure 3 in which it will be seen that accumulating the acceptable tolerance means that minimum or maximum durations for individual elements in the sequence cannot be specified.

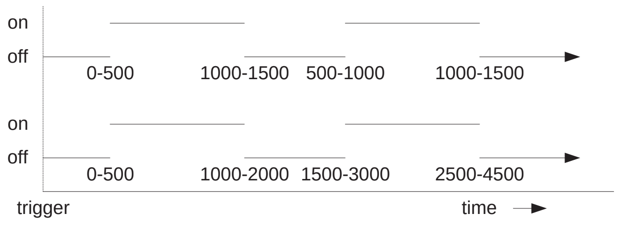

Figure 3: Specifying tolerances in timing step by step or from the initial trigger

One area where some difficulty attaches is when a timed intermittent behaviour is to continue for a set time, such as the belt minder chimer sounding intermittently for, say, five seconds. If there is a range of tolerance for timing of the intermittent soundings, then it might be possible that an extra chime be included if the chimer operates at the quick end of the range. One possible approach to this problem is to make use of the hierarchy to define the intermittent behaviour separately and have that incorporated in the timing. This implies the possibility of storing a library of functions (and / or post-condition behaviours) which would itself arguably be a useful addition to the existing language. Another possibility is the use of a timed exit operator for a cycle. There is one point about this which, while not really a problem, is perhaps worth making. This is that if the end of the intermittent behaviour happens to fall towards the end of an "off" period (so the chimer has been silent for a period) then the time of the intermittent behaviour might appear to be too short.

6 Dependencies between functions

Three areas where a function might be thought of as depending on another system function have been identified...

- Telltale or warning functions. These have been described at some length using the confusing table.
- Fault tolerant functions. While these are undoubtedly to be considered as separate functions from the fully working function, it is not clear to me that they really depend on the prime function of which they are the back up. They are separate because it is likely that the modeller will wish to identify cases where the prime function fails but the back up function works, such as a sensor failure meaning that ABS fails, but the braking still happens.
- Recharging functions, such as a cistern refilling after the flush. here is perhaps some room for discussion as to whether these are separate functions. The purpose is different. A toilet flushing is to clean the bowl while the refilling is to allow the toilet to flush again. Indeed, the purpose of any such function is to allow a repeat of the preceding function. Arguably this function is not called on if the prime function fails (assuming therefore that the tank is still full!).

One point to make early is that if the prime function is described in terms of pre- and post-conditions than these can be used as the preconditions for a telltale or warning function.

Another point about such functions is that they cannot be unambiguously described without preconditions, even in the case where these could be derived from a simulation of the correct system. This is because the triggering of such functions might depend on different incorrect behaviours.

The really interesting question about telltale or warning functions is that while they can sensibly be regarded as separate functions, with a distinct purpose, as well as pre- and post conditions, the purpose is not independent of the prime function. The purpose of any class of these functions can be summarised as "increase detectability of prime function". This has a clear knock on effect on the detection value for the prime function. Suppose we have a warning lamp that lights when current ceases to flow to the activator of the prime function for which it is a warning and that activator is switched on (so the prime function's preconditions are satisfied). A wire that connects with the activator going open circuit will cause the activator to stop but will result in the warning lamp lighting (depending where in the circuit it is) but a power failure might well affect both the prime function's and the warning function's systems so the activator fails and the warning lamp does not light. There seems to be a need for a different detection value for the prime function in each of these cases, so that the power failure has (sensibly) a higher RPN.

One possible approach to this problem is to have each function, with its own purpose, combined using AND to form a sort of über-function [1] with its own value for detection, which is that used if both functions fail, while the prime function's detection value is set assuming the warning function works correctly and is that used if the warning function does work. This is not inconsistent with the idea of descending the hierarchy of functions and subsidiary functions to find the appropriate severity value but does weaken the idea that the hierarchy of function

[1] I know 'über-something' is a horribly glib and trendy expression these days, but at least I didn't function with a k!

start with a specific purpose at the top. Note that this über-function does not actually have a purpose of its own.

The triggering of a recharging function actually depends on a side effect state (rather than a goal state) of the prime function. This is also one area where internal state is necessary, simply because as these functions are automatic, there is no need for this state to be apparent outside the system.

7 Limitations of the present work

The proposed approach is dependent on certain assumptions and limitations on the rôle of the functional description. I propose to list them here briefly, in the hope that this sparks discussion on how safe these assumptions are.

- Interpretation of simulation. The aim of this language is to provide a mapping between the output of a (behaviour based) simulator and some notion of a system's function, allowing the automatic generation of design analysis reports (such as for FMEA). It is therefore assumed that other knowledge is available for carrying out the simulation.

- There is no need top represent "internal" function, such as component level function. Function is used as a mapping between behaviour (internal) and purpose (external). As such it depends on the state at the edge of the system (or of the simulation) - the system properties with which it is concerned are those that have a direct affect on the system's environment. It seems to me at least arguable that for interpretation there is nothing to be gained by explicit representation of internal function, that information is captured by the the knowledge of structure and behaviour on which the simulation depends. For failure analysis, indeed, component level function might be seen to cloud the issue by requiring functions attaching to the failure mode behaviours. Does a wire have "conduct to ground" as a function resulting from the short to ground failure mode? Presumably any definition of function that captures the idea of intention excludes this? This is one area where functional knowledge might be affected by the design analysis task. There is clearly a case to be made for specifying component function for design verification, if only as a shortcut in establishing how a design fails to meet its requirements.

- The system and the simulated world amount to the same thing. It seems arguable that any need for interpretation as a separate activity from simulation depends on this assumption. An ideal simulator might be able to derive a good deal of knowledge of purpose of a system from its knowledge of the system and the world. For example, it might know enough to derive the fact that a current flowing through a certain resistance will cause the wire to glow (as it is in a bulb) and that it is powerful enough to shine a beam (it knows about the reflector) and it is not impossible to accept that it also knows that the lamp is on the front of a car so will light the road ahead. If it also knows something about traffic law it will also appreciate that another such lamp is needed. Dream on... The point I am driving at is that asking the user to define such a mapping is in a sense an attempt to meet the limitations of the simulator, in the same way as telling it that current through a lamp causes it to be lit or that the abstract states in (Keuneke and Allemang 1988) sidestep the dependence of interpretation on repeated behaviour.

All these points listed are closely related, indeed I suspect that they all more or less follow from one another, and the central assumption that a functional description language that describes the behaviour (state, output) of a system that directly affects its (un-simulated) environment is sufficient for interpretation of simulation of a system based on knowledge of its structure and (component) behaviour.

8 Questions

As usual for these reports, I shall finish with some questions or areas for discussion rather than a conclusion.

- Are the assumptions and limitations suggested above sensible? Are they unduly restrictive?
- How does this fit in with the relationships between behaviour, function and purpose? I think fairly well, but there is room for more work on this. There is another report which looks at this issue from this point of view, addressing (or attempting to address) these more pedantic questions.
- To what extent does the design analysis task affect the usefulness of this approach? Are internal functions more interesting for design verification, for example?
- Is the idea that a function fulfils one purpose sensible?

References

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.

Pugh, D., C. J. Price, and N. A. Snooke (1995). Practical applications for multiple models - the need for simplicity and reusability. *Applications and Innovations in Expert Systems III*, 277–291.

Snooke, N. A. (1998). Hierarchical functional reasoning. *Knowledge-Based Systems 11*, 301–309.