Functional modelling for SoftFMEA

Jon Bell

Doc. ref. SD/TR/FR/01: June 17, 2003

1 Introduction

This report discusses functional modelling in AutoSteve, specifically its limitations with respect to SoftFMEA, and it will introduce possible approaches to solving the resultant problems. Recent discussions have resulted in a set of questions for SoftFMEA to consider. These are listed in [2]. Many of these are concerned with behavioural modelling but some are more concerned with the functional modelling side of AutoSteve. The need to start a thread of research into functional modelling is also raised in [2], as are some possible answers to the questions listed therein. This report will expand considerably on the material on functional reasoning in that report.

Some familiarity with AutoSteve is assumed, however Section 2 will briefly discuss functional modelling as applied to AutoSteve. Two main areas where the existing arrangements for functional modelling seem insufficient have been identified, each area is discussed in the following sections. These are problems associated with timing, discussed in section 3 and function dependent on complex behaviour, section 4. Section 5 will introduce and discuss possible solutions, including alternatives where they have been identified. A following section (section 6) will then briefly discuss functional modelling problems associated with fault mitigation.

2 Functional modelling in AutoSteve

It might well be argued (especially by those reading this report!) that an internal report is not the place for a lengthy background discussion of functional modelling and its place in the AutoSteve system. However, it is felt that a brief discussion is worthwhile for two reasons. The first is to give readers the opportunity to correct any misapprehensions about AutoSteve functional modelling which might lead to poor arguments in this report and the second is simply that as this report is likely to inform later, more extended works that will require such material, some discussion here will be useful in future.

Chittaro and Kumar, in [3] list functional knowledge as one of four useful classes of knowledge in model based reasoning. These four classes of knowledge are: -

- Structural — what is in the system and how are these elements connected?

- Behavioural — how does each element work?
- Functional — what does each element do?
- Teleological — what is each element for?

It can be argued that in a given case, not all these classes of knowledge need be explicitly represented. For example, Hawkins and Woollons, [4], use functional modelling of their components as the dynamic aspect of their simulation, so have no need of an explicit representation of components' behaviour.

AutoSteve takes the opposite approach and uses a behavioural representation of components in simulation and relates function to the system. There is therefore (arguably) overlap between function and teleology. Components have no individual representation of function, function instead being treated as an aspect of the system, mapping between the system's purpose and component behaviour. This is an example of the "associational" representation of function, as distinct from the definitional representation used by Hawkins and Woollons. This simply means that function relates both to behaviour and teleology while a definitional representation of function will describe the function in terms of predefined primitives.

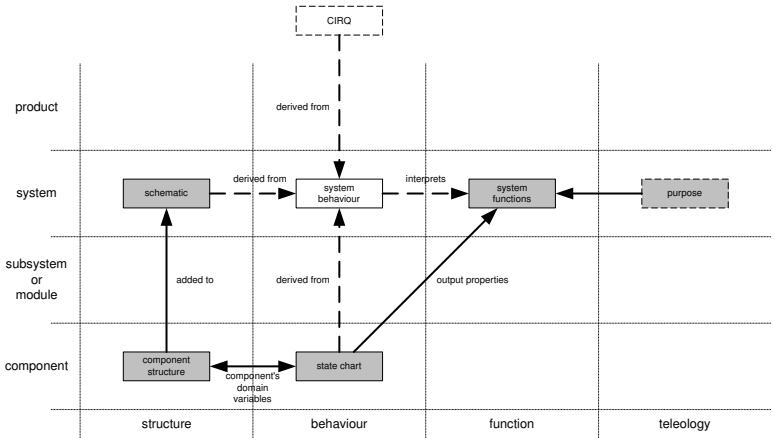

Figure 1: The models used by AutoSteve

Figure 1 shows the models used by AutoSteve and the relationships between them. Component behaviour is mapped to system function using component "output properties". A lamp, for example, will have the output property "glowing" which might simply be associated with the state (behaviour) of having current flowing through the filament. It might be suggested that this output property is properly a mapping between behaviour and function but it has been found that this mapping is so simple that there is little benefit in an explicit functional model of a component. Associating a generic lamp function (call it "lit") to the output property adds no information. It fails to relate the property to purpose, as the purpose of the lamp

will depend on the system, and indeed on its role in the system. For example, a stop lamp has a different purpose from the headlamp of a car. As the lamps themselves might be identical (qualitatively), using the output property in preference to a teleological function aids component re-use. This is discussed in [6]. It should be noted that while this is true with respect to the component, it is not true with respect to the system. There is nothing to prevent a user from associating a function with a single component, and using these subsidiary functions to build up the full system function, as suggested by Neal Snooke in [7]. However, AutoSteve does not allow the user to explicitly link a function to a component as opposed to a system. The idea of a hierarchy of functions suggested in [7] uses logical operators (AND and OR) to combine the subsidiary functions. In practice it is necessary to attach each component's output to a its own function. This is because in running a failure mode and effects analysis (FMEA), the results are expressed in terms of achievement of (system) functions. Therefore if, say, a car lighting system's sidelights on function is linked directly to the two sidelights' output property, then when one lamp fails the function will, correctly not be achieved. However, it is also necessary to find unexpected achievement of a function, so if a failure causes one of the lamps to remain lit whatever the switch position, this will not result in unexpected achievement of the sidelights on function as it is never achieved unexpectedly, being dependent on both lamps being lit. If the sidelights on function uses two subsidiary functions, right sidelight on and left sidelight on, then if, say, it is the left bulb that is on all the time then the left sidelight on function will be achieved unexpectedly and this will be shown in the FMEA report.

Function is a relational concept — it defines the relationship between behaviour and purpose and may also help define the relation between a component and its system, as outlined above. Another important relationship defined by function is that between input and output. This is clearly more important in an operational function (such as in [4]) but also applies to the purposive functions used in AutoSteve. However, when setting up a system's functional model, input is not explicitly specified in AutoSteve. Instead, in FMEA, the correct system inputs for a function (that is, the switch settings when the function should be achieved) are derived from the initial simulation, when the system is simulated with no component failures. This further simplifies functional modelling, but means that design verification is not done, the idea that correct achievement of system functions can be derived from correct operation of the system implies the design is also correct. It should be pointed out that the user might specify the correct inputs associated with a function as they are used in Sneak Circuit Analysis.

The Aberystwyth approach associates function with the system and behaviour with components. This contrasts with the approach taken by Sticklen et al [8] where behaviour is associated with a function, describing how a given system function is to be achieved. This relationship is shown in figure 2. This approach places more emphasis on the user providing the specific models required for analysis, while the models required by AutoSteve are more readily available. The component behavioural models required by AutoSteve are more generic, as the behaviour is not

explicitly related to system function. This makes them more reusable.

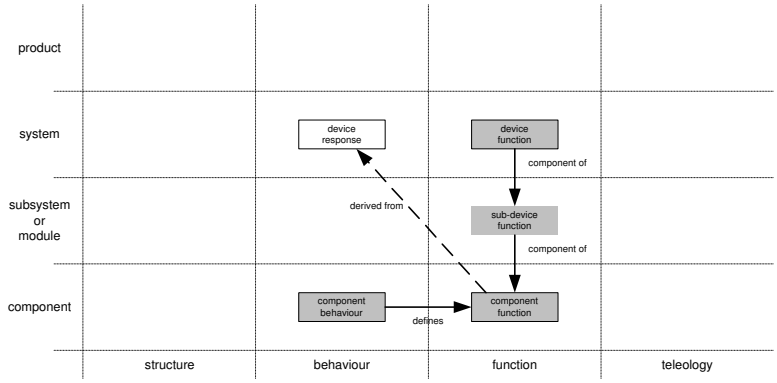

Figure 2: The models used by Sticklen and others

It will be seen from the foregoing that there has been a consistent tendency to simplify functional modelling in the development of AutoSteve — component functional models are not used, and even the inputs to system functions are derived automatically on running an FMEA. Unfortunately, while this simplicity is desirable, it causes problems when modelling the kind of complex systems that are the subject of the present project. These problems will be described in turn, and possible approaches to their solution introduced.

3 Timing and function

There are two aspects of timing that might impinge on functional modelling. One is where the timing of the output is significant, so a function might be achieved late and the other is the timing of input, so an input might have distinct effects, depending on when it takes place. These aspects are to be expected if function is a mapping between input and output. They are each discussed in their own subsection.

3.1 Late achievement of functions

With the increasing use of networks in vehicles, the possibility arises of a system's operation being delayed by late transmission of a network message. This is especially true where CSMA/CR protocols such as CAN are used, where low priority messages might have to be transmitted several times before they are received, previous transmissions having been lost through arbitration resulting from a message collision.

Detecting late achievement of a function requires more explicit coupling between input and required output. While at present the function "headlamps dipped" might be defined simply in terms of output, so it is achieved whenever both the left headlamp and right headlamp are on dipped, if we are to capture late achievement of

the function we need a deadline. This must be measured from the time the required input has been entered, so the required input arguably needs to be specified. In practice this need not be the case, as the correct input can still be derived from the simulation with no component failures. All that would be needed is some way of setting the deadline itself.

AutoSteve can simulate using a qualitative or quantitative notion of time, either could be used to specify the deadline. The simulation with no failures cannot be used to establish this deadline, as this would mean that any increase in time to achieve the function, no matter how insignificant, would result in late achievement of the function. It is possible that relying on the initial simulation might be acceptable if qualitative time was used in setting the deadline. If the correct simulation had the function achieved in milliseconds and a failure resulted in it only being achieved in seconds, this could be regarded as being sufficient to regard achievement as late. However, even in this case, if a function was achieved instantly in the correct simulation and only achieved after microseconds because of some failure it might well be the case that the delay is not significant, despite the order of magnitude difference in the time slots in which the function was achieved. Also, of course, relying on such large orders of magnitude might make a significant delay undetectable — correctly the function might be achieved in say 25 milliseconds while a failure that results in it not being achieved for 500 milliseconds might well be beyond the deadline, despite the delay being in the same order of magnitude time slot.

While all the foregoing is conceptually quite simple, it does lead to problems. The most important of these is the need to capture inputs where a component has memory. For example, it is easy to model a dipped headlamps function where it depends on, say, the dip switch position. It is a good deal less straightforward if the dip switch is a toggle switch, so pressing and releasing it results in the states swapping — the new state depends on the old state. This might imply that we should be prepared to map function to state rather than simply to input. Can this lead to problems if such a component has failure modes that alter the relationship between inputs and states? It certainly reduces the neat implicit mapping between system inputs and outputs and function. Possible solutions to this problem are introduced in section 5.

3.2 Timing of input events

One aspect of complex systems that has not had much consideration so far is the significance of timing of input events. For example, the "belt minder" case study used as an example in section 4.1 allows the driver temporarily to disable the warning system if he unbuckles his seat belt within a certain time (three seconds). This, of course, means that the same user input has different effects, depending on the time that has passed since the previous input. There is currently no way of capturing this information.

This requires some refinement of the scenario and simulation tools to allow the timing to be captured. How this is captured requires more thought. It should

not be attached to the component as it is an aspect of the system behaviour, not the component behaviour. It therefore seems reasonable to suppose that it will be attached to a system model. There is more on this in the subsections on possible approaches, sections 5.1 and 5.2.

4 Function and complex behaviour

Two aspects of system function being dependent on a complex behaviour have been identified — where a function depends on a complex behaviour to be correctly fulfilled and where a system function is apparently a steady state, but more detailed modelling of the behaviour reveals more complexity, such as a cycle. This latter case is felt not really to be concerned with functional modelling but has been included briefly for completeness.

4.1 Functions dependent on complex behaviour

Recent work on a case study (the "belt minder" system) has shown that modern systems might well require system function to be mapped to complex behaviour, dependent on the correct relationship between different components behaving correctly.

The purpose of the belt minder system is to warn the driver of a car if he drives off with his seat belt unbuckled or his front seat passenger's seat belt unbuckled, if there is a front seat passenger. There are two warnings. A dashboard warning light will come on and stay on and a chimer will sound intermittently for a period of time before stopping.

Correct functional modelling of the warning function needs to be associated with correct behaviour of both warnings. The lamp, as it comes on and stays on, is simple. The chimer is more complicated and cannot readily be modelled with the existing functional modelling, though the use of the sequence feature might help here. This is made worse by the fact that AutoSteve simulation continues until a steady state is reached, when the functions are checked. This means that the functional model is only used once the chimer has stopped working, so it should be off. In figure 3, the simulation will stop once it reaches the state `time_is_out`, when the chimer will not be sounding.

If we are to model the chimer as a simple electric buzzer that sounds when current flows through it and is otherwise silent, then the control behaviour necessary for its intermittent sounding will come from some other component. Therefore correct achievement of the warning function must depend on correct achievement of a chimer sounding function that in turn depends both on the chimer going off correctly and on its being correctly controlled by its controller. It is worth pointing out that simply combining these two behaviours using logical AND is insufficient, as both components might work correctly, but a fault in the connection might means one is not driving the other. Therefore the function needs to be based on the output, not simply the component behaviours.

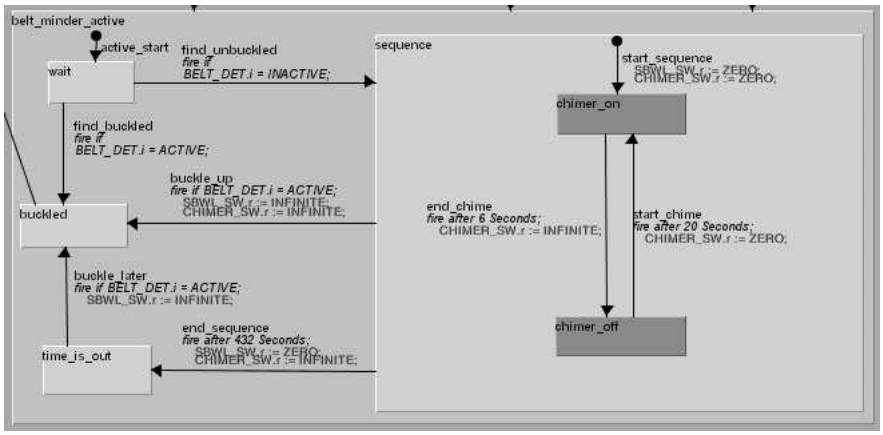

Figure 3: The behaviour of the chimer controller when the warning is sounding.

It is possible that the existing sequence facility int eh functional modelling tool is sufficient here, but it needs to be established that it is so reliably and in all cases. It might well need some refinement, as it will be worth capturing the timing of the sequence of events. This needs investigation.

4.2 Cyclical behaviours

One other aspect of behavioural complexity which warrants some consideration is cases where an apparently steady state is actually a cyclical behaviour. One example might be a lighting system in which the lights are switched on automatically triggered by input from a sensor. In this case, a likely behaviour is that the sensor periodically sends its current reading to the controller, even though in most cases, this will result in no change of state.

This is arguably a behavioural modelling problem rather than a functional modelling one. After all, there is no need to define the periodic sending of sensor readings in terms of system teleology. It has been included here for the sake of completeness. Indeed, it could be argued that attempting to capture this detailed information in a functional model leads to a reduction in the re-usability of the functional model. For example, there is no need for this model to show that the sensor periodically sends its reading, as the same functional model could be used with a more intelligent sensor that did only send a reading hen the change was sufficient to result in a change of function.

This case suggests that while the in situation outlined above (in section 4.1), the function depends on the complex behaviour, in this case the function does not depend on the complex behaviour. This supports the idea that this is a behavioural modelling question rather than a functional modelling one. For completeness, there is material on this in the sections on possible approaches, sections 5.1 and 5.2.

It is worth noting that there is a more complex case related to this, where there

is feedback in the system. A previous report, [1] has dealt with this, and it is not considered further here.

A further aspect of this, that is not covered here, is the need for the simulator to identify a cyclical behaviour and stop the simulation once one has been identified. This is clearly a behavioural modelling and simulation problem rather than a functional modelling problem.

5 Possible approaches

The sections above, Sections 3, 4, suggest a need for a more sophisticated functional model. Various approaches to building this model are discussed below.

Because of the need to express input more explicitly, to allow deadlines to be set in modelling late achievement of functions, the existing functional model is too simple. Simply adding the right input to a function is not possible using this functional model, as in some cases the effect of the input will depend on the current state of the system, such as when a toggle switch is used to switch between desired functions. We therefore need to add some sort of representation of system behaviour to the functional model. This should also help with describing functions that depend on complex behaviour. Two possible approaches to this are described below.

5.1 Use the FMEA scenario

While the complex behaviour within a single function might be capable of being modelled using the sequence facility in the existing function builder, the need to capture inputs that depend on a previous state of the system, such as the toggling of the dip switch, mean that some means is needed of relating input to system state.

One simple approach to this is to use the existing scenario editor. An FMEA is specified by a scenario that specifies what inputs are to be tried, and in what sequence. AutoSteve already has a user interface component that allows these inputs to be specified, and it would be quite straightforward to allow this tool to be used to let the user specify the expected functions associated with each successive input. Therefore once the headlamps have been switched on and are dipped, the next press action on the dip switch should result in the system switching to main beam. It would not be difficult to specify a deadline at the same time, to allow late achievement of a function to be detected.

This approach has the benefit of simplicity, and of demanding little extra work from the user. However there are drawbacks.

The most important of these, from a theoretical stand point, is that the functional model is dependent on the scenario. Therefore, if the scenario is not complete, neither will be the functional model, and some functions might remain untested. As a side effect of the functional model being dependent on the scenario, it cannot readily be reused. There will, it is supposed, be a similar functional model to the existing one that can be re-used as readily as at present, all extra features being attached to the scenario. This does not answer the problem that deadlines for

achievement of a function are attached to the scenario, so if different scenarios are used, the user will need to ensure that these aspects are consistent.

A related problem to this is that the functional model will not be available without running the scenario. Therefore it will not be available when running a step by step simulation.

This approach also relies on the ability of a function to capture complex system behaviour, no additional functionality is added here. This might well not be a problem, provided the sequence facility is both reliable and sufficient in all cases. This remains untested at the time of writing, this is an early task in investigation into functional modelling.

This approach is well suited to the idea of capturing the timing of inputs in the scenario, of course. Indeed as this is an aspect of the scenario, rather than the functional model, some way of adding the idea of timing to inputs is necessary whatever approach is taken.

As no support is offered to capturing complex behaviours in a function, this approach does not help with identifying a function with a cyclical behaviour.

5.2 System functional model

In view of the drawbacks with the approach outlined above, a better approach might be to construct a functional model that specifies the required behaviour of the system. Such a model for a simple running lights system might look like figure 4. It could be argued that this is a species of requirement specification in that it

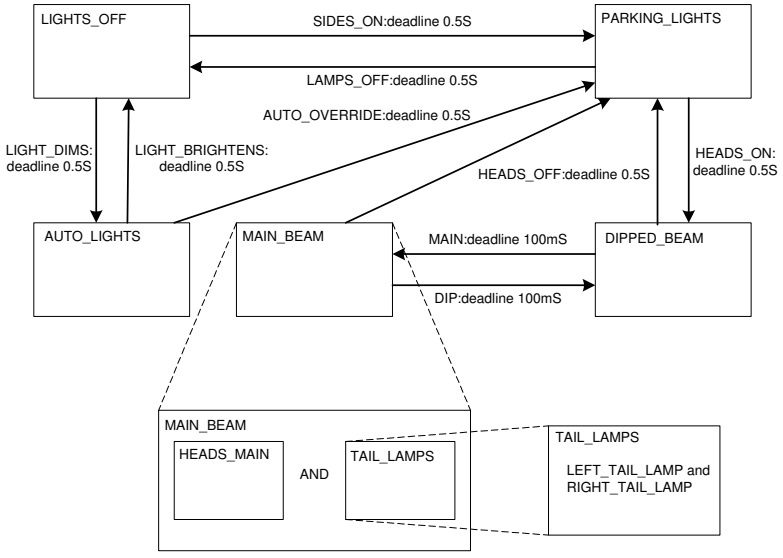

Figure 4: Required functions for a lighting system.

captures the intended behaviour of the system. In other words it shows what the

designer intends should happen, not what does happen. For example, the diagram specifies that the system is to have an input that allows the parking lights to be switched on when the lamps are off, but that there is no need to switch directly between off and headlamps on. This, of course, allows the use of a rotary switch. The diagram therefore maps between purpose and the required (expected) system behaviour, perhaps a fairly complex representation of teleology. The functional hierarchy for the tail lamp function has been included as an example of how the top level functions should map down to individual component functions that in turn will map to component outputs. The required inputs will also map to component inputs. How far down the hierarchy the suggested graphical notation is used might depend on individual cases. Here, each function will simply be a logical combination of lower level functions, as shown for main beam, and so there is perhaps no need for the graphical notation below this top level. However the warning function in the belt minder system discussed in section 4 might involve behaviour of sufficient complexity that a graphical notation might be helpful. There would seem to be no problem in having a graphical notation, similar to a state chart to describe behaviour to be associated with a function. The use of a functional hierarchy is not new, of course, see [7]. What is new is the expression of required behaviour in the functional model, a mapping between the top level functions and inputs. As an aside, it is worth pointing out that we might enforce a 1:1 mapping between function and output by not allowing the use of logical AND between component output properties when defining a bottom level function.

There is no reference here to how the required inputs are generated. The light switch could be a three position rotary switch on the dashboard and the dip switch separate, such a s a stalk on the steering column or both switches might be the same physical component. There is also an implicit need for a sensor to detect the ambient light level. This supports the idea that the functional model is reusable for any lighting system with this behaviour, it does not depend on the actual system. For example, all it demands is an input switching between dipped and main beam - the dip switch could be a toggle switch or have distinct positions for the two functional states. How far the model can be reused is interesting, as it depends how far down the hierarchy we try to use the model. For example, the top level functional diagram could be used for a motorbike, but the idea that, say, the tail lamp function contains left tail lamp and right tail lamp is not reusable in this case. As reusability of the functional model is an important advantage of this approach over the scenario based approach introduced in section 5.1 this question needs addressing. A suggested solution is to include all system functions in this model, merely mapping these to actual outputs and inputs on attaching the functional model to an actual system. This would, of course prevent the model being reused where the functional hierarchy differs, as in the motorbike example.

As the functional model does not specify how the inputs are generated, some way of attaching actual inputs to the functional model's "abstract inputs" is needed. A suggested approach here is to "instantiate" the top level functional diagram, replacing the generic input labels with actual system input properties, as in figure

5. The modelling here assumes the use of a toggle switch for dipping the headlights

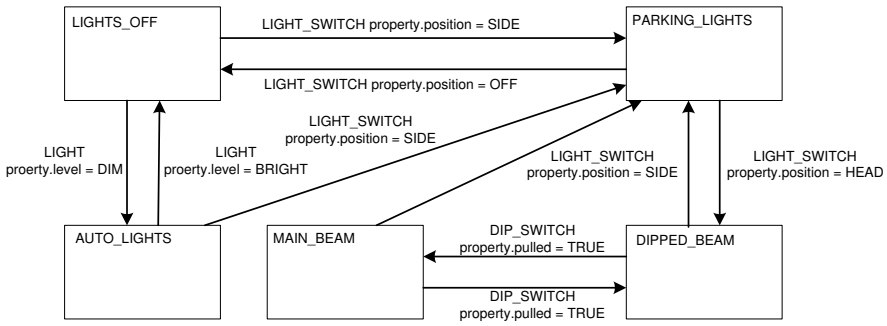

Figure 5: The lighting functions, attached to a specific system

and also that the light level is an input property of a sensor component. The main problem is capturing the idea that the dip switch being pulled are separate events — the system will not cycle between main beam and dipped while the switch is pulled on! This way of modelling the sensor is adequate in this case, there would be no difficulty in modelling failure mode behaviours that failed to transmit the required change of system state to achieve the auto on function.

The most important disadvantage of this approach is the additional work asked of the user. It is appreciated that drawing a top level functional diagram for the belt minder system will not be straightforward! However, once the functional model has been generated, mapping it to actual system inputs should be straightforward.

A compensatory advantage is that there seems to be a possibility of using this diagram to automatically generate a suggested scenario. One difficulty here is that the meaning of the diagram is not unambiguous. For example it specifies an input from off to parking lights on, but does not explicitly rule out an input leading to switching from lights off to dipped headlights. Of course, it would be quite possible to take the meaning that all intended functional changes are included in the diagram, so other changes need not be tested for. The difficulty of automatically generating a scenario centres on this point - should it try unspecified changes (where an input is available) to ensure that unintended effects are not found? A related problem is constraining behaviour of input components. A scenario might, for example, attempt to switch between, say auto on and headlights on by switching directly from light switch off to light switch heads, but if the light switch is a three position rotary switch, this transition is impossible.

As has been noted above, when running an FMEA, the timing of input events is clearly an attribute of the scenario, not the functional model, so some means of adding this information is needed in the scenario tool. The more detailed functional model, will show the need for such timed events, but that is all it will do. Such events will, of course, want to be included in any automatically generated scenario.

While it is supposed that cyclical behaviours will not necessarily be specified in

the functional model, as discussed in Section 4.2, the graphical notation could be used if such behaviour was to be specified.

It is perhaps worth adding a brief description of how this extra model might be incorporated into the use of the tool. As was implied by the suggestion that it amounted to a sort of requirements specification, it is imagined that the functional model would be the first model created in developing a new system. Obviously this will not be done if an existing functional model can be re-used, or an existing functional model could conceivably be copied and modified (for example if minor changes such as deadlines for achievement of functions) were all that was to be changed. Once the system had been drawn and components' input and output properties specified, these could readily be mapped to the functional model, much as is done at present. The functional model can then be used to inform the generation of a scenario that tests all the necessary system behaviour and functionality. In future, as suggested above, it might be possible to automatically generate a suggested scenario, which the user could modify if necessary, using the existing scenario editor.

This approach has a good deal in common with the earlier work on design verification, see [5]. Indeed it is quite possible that the proposed functional requirements diagram could be used for design verification. The design verification work used an attainable envisionment to generate a state chart of the system's actual behaviour that could be compared with the required behaviour.

5.3 On capturing system behaviour

Both the alternative approaches described above are concerned with capturing the intended system behaviour (that is a more elaborate teleological model of the system, not a behavioural model of the whole system). There is no need for a behavioural model of the system to be made by the user — that is the job of the simulator. As the simulator alternates between structural and behavioural simulation, it builds up a step by step picture of the system's response to an input. This can be compared with the functional model's required behaviour to establish whether or not the required function has been achieved.

In the case of the scenario-based functional specification, this will need to use the existing sequence functional specification (or some suitable refinement of it), or will need to compare the behaviour resulting from an input during a failure mode simulation with the correct (no failure) simulation.

The complex functional model will allow the user to specify the expected behaviour for a function to be achieved, and the simulation's behaviour can be compared with this.

6 Fault mitigation and functional modelling

It should perhaps be pointed out that as yet, little thought has been given to fault mitigation, but it is to be considered later in the project. It seems reasonable

to introduce possible problems this might give rise to with respect to functional modelling, and to speculate on possible approaches.

The major question here seems to be whether fault tolerant behaviour gives rise to some sort of alternative function. For example, an electronic ignition system will adjust the timing of the spark to suit the current conditions under which the engine is running, but in the absence of input from some sensor might resort to a default approach to timing. The question is whether this can or should be regarded as a distinct function, which can readily be included as such in the top level functional diagram. It seems reasonable that it should, as this fits well with the functional diagram's presumed rôle in requirements capture. It is imagined that the required fault tolerant behaviour will be specified as a requirement. It is appreciated that drawing such a diagram for a system which is to have much complexity in its fault mitigation strategy will not be easy but if we are to accurately simulate a system, we need an accurate model — the more complex the system, the more complex will the model be.

The scenario approach outlined above (in section 5.1) does not readily allow any fault mitigation behaviour or functionality to be specified. This is because it only allows the user to relate input to function, not input and failure to function. It is appreciated that it would be possible, if inconvenient, to allow the user to specify the expected output given both the input and failure but this would be extremely cumbersome.

The functional model described in section 4 could allow the user to add specific fault mitigation functions, reached only when there is a failure. These might either be added to the same diagram, or there might be a separate diagram that specifies the system behaviour when there is a specific fault, in a similar way to the possible use of a separate state chart if a component's failure mode behaviour needs a description. Which of these two alternative is preferable needs more thought.

7 Conclusion

This report has set out two areas in which the current AutoSteve functional model appears to be inadequate for the types of system SoftFMEA might be expected to model. Two possible solutions are proposed. Of these the functional diagram approach seems more "correct" in that the notion of function is kept more independent. It is, perhaps, an open question whether this correctness justifies the greater elaboration in modelling. At the time of writing this draft, there is room for more work on how well the functional diagram works with complex functions.

Underlying this report is the relationship between models in AutoSteve. This has so far been kept quite simple and well defined, however the complexity of modern systems appears to militate against these qualities. This is seen at both the behavioural and functional levels. At the behavioural level we now have components behavioural models affecting each other with no intervening structural simulation, although, of course, it is important that the structural model provides a suitable con-

duit for this message passing between components' behavioural models. An example is the need for a physical connection along which CAN messages can be passed. At the functional level we might wish to describe functions whose correct achievement cannot simply be defined in terms of a single component's output, such as the chimer output in the belt minder system. This means we need both a more complex functional model (specification) that describes the required behaviour and some means of comparing this specified behaviour to the actual behaviour of the system.

One important feature of the AutoSteve modelling is the clearly constrained relationship between the different models, both at the component and system level, as shown in figure 1. We need to preserve this as fully as we can. Some principles we should try to maintain are: -

- All connections between components are defined in the structural model. This might need to be weakened in the case of sensors that detect the effect of the output of another component.
- No component should know how any other component behaves. It can, of course, know what outputs another component can give, provided there is some means by which this output can be detected or received by the component.
- The functional model does not know how the desired outputs are achieved. For example, the belt minder functional model does not specify that the GEM generates the intermittent behaviour for the chimer.

Two approaches have been discussed. The idea of using the scenario is simple and demands little extra input from the user. However its problems suggest that the more elaborate (and correct) functional model approach is to be preferred. That approaches offers advantages, apart from its correctness, including better support for modelling functions that depend on complex behaviour, greater re-usability, and better support for modelling fault tolerant functionality.

References

[1] Jon Bell. The heater circuit - matters arising. SoftFMEA internal report, 2002.

[2] Jon Bell. Softfmea: Questions, approaches and answers. SoftFMEA document ref. SD/TR/GEN/01, 2003.

[3] Luca Chittaro and Amruth N. Kumar. Reasoning about function and its applications to engineering. *Artificial Intelligence in Engineering*, 12(4):331, 1998.

[4] P. G. Hawkins and D. J. Woollons. Failure modes and effects analysis of complex engineering systems using functional models. *Artificial Intelligence in Engineering*, 12(4):375, 1998.

[5] Alex McManus, Christopher Price, Neal Snooke, and Richard Joseph. Design verification of automotive electrical circuits. In *Proceedings 13th International Workshop on Qualitative Reasoning*, Loch Awe, Scotland, 1999.

[6] Christopher J. Price. Function-directed electrical design analysis. *Artificial Intelligence in Engineering*, 12(4):445–456, 1998.

[7] Neal Snooke. Hierarchical functional reasoning. *Knowledge-Based Systems*, 11:301–309, 1998.

[8] Jon Sticklen, A. Goel, B. Chandrasekaran, and W. E. Bond. Functional reasoning for design and diagnosis. In *Proceedings Model Based Diagnosis International Workshop (DX-89)*, 1989.