Evaluation of AutoSteve data bus support

Jon Bell

Doc. ref. SD/TR/EV/01: October 23, 2003

CONFIDENTIAL

1 Introduction

This report will present an evaluation of the data bus support added to AutoSteve from version 2.6. This addition to AutoSteve has arguably rendered the need to implement software for modelling electrical failures in systems that incorporate network components redundant, so this evaluation can be considered an alternative approach to fulfilling SoftFMEA Workpackage 3 in the original project plan, described in (Snooke and Price 2000). As there is a further workpackage concerned with extending this to model failures in the software or network parts of these systems (Workpackage 4 in (Snooke and Price 2000)), this report will also discuss the AutoSteve data bus support in terms of how well it meets these requirements and what further extensions might be required for modelling of software or network failures.

Evaluation was started using version 2.6, but the tests carried out have since been redone, and work continued, using the recently installed version 3.1.

1.1 Scope of this report

This report is concerned specifically with the modelling of systems that incorporate data bus components, allowing separate components to communicate digitally. There is no discussion of the modelling of individual components that themselves use software. This is inevitable given the document's focus on support for modelling data buses, and follows from the project's early focus on such networks, especially CANbus. As a result of this focus, this report is concerned with how well the data bus support facilities added to AutoSteve support modelling of CAN (and similar protocols). In practice this is quite an appropriate approach to take as CAN is a CSMA/CD protocol, so message collisions are possible, and message latency is not deterministic. It is therefore likely that if CAN is supported, the same bus support can be used to model time-triggered networks, as these difficulties in modelling should be avoided.

In view of the report's discussion of AutoSteve, familiarity with the software is assumed. Familiarity with CAN and other network protocols referred to is also assumed. CAN is described in (Bell 2001b) and other protocols used in the automotive industry are described in (Bell 2001c). These protocols are compared in (Bell 2001a).

1.2 Structure of the report

The main purpose of this report is to comment on the findings of work on evaluating the bus support for modelling CAN. This has been arranged in one section, Section 2, subdivided into subsections for each area for discussion. As it seems worth adding material on what might be needed to allow failures in the network to be modelled, this has been added as a further section, Section 3, following the evaluation material.

1.3 Background

As stated in the introduction above, this report in concerned with how well the data bus support added to AutoSteve from version 2.6 can be used to model automotive electrical systems that make use of a data bus, specifically one using the CANbus protocol.

Because we have found few useful case studies, the work is largely based around systems devised here. These have been kept simple for a variety of reasons. These include:-

- The desire to confine modelling problems to the network components. I wanted to avoid delays and distractions resulting from attempting to solve modelling problems not arising from the network itself.
- Difficulties arising from modelling systems whose correct functionality depends on complex behaviour, specifically cases where the function depends on temporal relationships between system outputs. These problems have been discussed in (Bell 2003b).

2 Evaluation of existing bus support

For the reasons outlined in Section 1.3, much of the evaluation work was done using simple systems devised especially for the work. One such system that was used for much of the work is a version of the old simple lighting circuit, modified to use a data bus, illustrated in figure 1. While the system is not an accurate rendition of any actual system, it does have features taken from a Jaguar system. Apart from the obvious one (the use of a network), the most important feature taken from the Jaguar circuit is the addition of the "autosensor" that is assumed to switch on the dipped headlamps once the light falls below some preset threshold. In the case of this model it is, of course, merely another switch, with the light threshold set as an input property. It is valuable for the current work, as it is possible to allow both

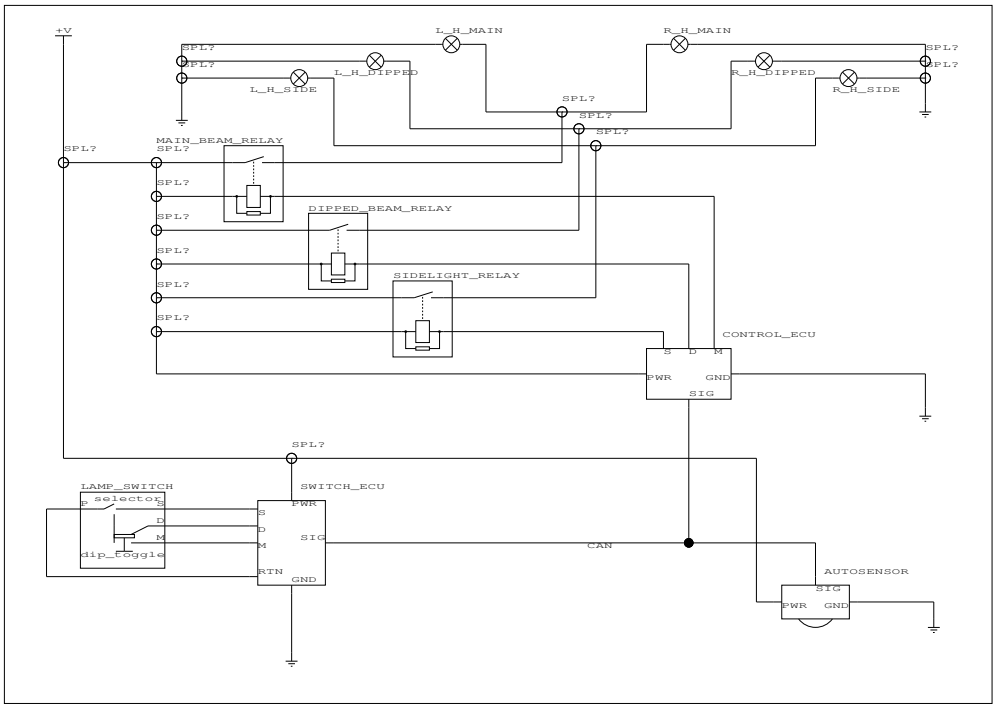

Figure 1: A simple lighting circuit using CAN

the autosensor and the switch to send messages at the same step in the simulation, to generate collisions between messages from the ECU and the autosensor. It will be appreciated that this a poor model of a sensor's likely behaviour. It is likely that a sensor will send a reading periodically and it will be left to the lighting control ECU (in this case) to interpret the reading. The way the system has been modelled has placed the interpretation in the hands of the user, expressed using the sensor component. The question of the different transmission types (such as periodic transmission) is considered in Section 2.5 below.

The tests have been done using the simulator, rather than running FMEAs. This was because it was felt necessary to examine the behaviour of the system in more detail than seemed convenient from an FMEA — it was felt that the functional model's interpretation of the behaviour would not be helpful, and it was desired to examine the state of the system after individual input events. I appreciate that this can be done from an FMEA, but it seemed simpler to use the simulation tool.

The data bus support is intended to allow different protocols to be modelled. These tests are concerned with CAN, so I have not looked into facilities for point to point communication, though this is supported. In the systems modelled, this is hardly relevant, but it should be possible to have a component ignore messages unless the id is appropriate (i.e. it can reject messages according to the message id). This would approximate to modelling so-called "full CAN".

2.1 Identifying messages

It was noted above that it should be possible to identify messages by their id and reject those that are irrelevant to the receiving component. If the "message" data type is used, this can indeed be done. No attempt has been made to model rejected messages but in the test circuit the control ECU needs to distinguish between the two senders, as the same value has a different meaning in the two messages. This it does correctly, so there is no reason to suppose that a component cannot be made to reject messages according to the id. Indeed, this already happens in the test circuit as messages from the autosensor are ignored if the lights have been turned on by the driver.

This would allow the possibility of modelling the CAN terminal as a separate component form its associated ECU and mean that only the relevant messages were passed to the ECU. This will presumably only be worth doing if the CAN terminal and ECU are electrically distinct (so they have different connections to the power supply, say) or if enough of the network is being modelled that irrelevant messages are likely. Another advantage might be to simplify modelling of complex behaviours, by separating them into distinct components. This can, perhaps better be handled by using separate concurrency groups for the CAN behaviour ad the ECU behaviour. This has been tried and seems to work fairly well, but might not be a perfectly accurate model of the actual behaviour. The possibility would exist, perhaps, of having a configurable CAN terminal component that merely needs configuring with the ids of the messages it should pass on to its ECU on receipt.

2.2 Generating a collision

One test felt necessary was to ensure that the collision modelling provided worked. When using the AutoSteve bus support, if two messages are sent simultaneously, they will collide, unless they have priorities set, in which case the higher priority message will be transmitted.

One interesting problem that was identified here was that the collision depends on both messages being generated in the same way. Using the simple lighting circuit in figure 1, attempting to generate a message collision when having the autosensor and switch ECU sending messages at the same step of simulation failed consistently. Instead each message was written to the bus correctly, but only the first one was read. This meant that the simulation finished with the dipped headlights on when it should have finished with the sidelights on.

It was found that if the expected collisions were to be generated, it was necessary to eliminate the electrical simulation that was necessary to drive the change of state in the switch ECU. This was done by combining it with the switch itself, as shown in figure 2. This is not an appropriate solution because the wires connecting the switch and the ECU are also lost and so will their failures be in FMEA. One possible work around is to model the switch and ECU as separate components but have them communicate by passing messages rather than by using an intermediate

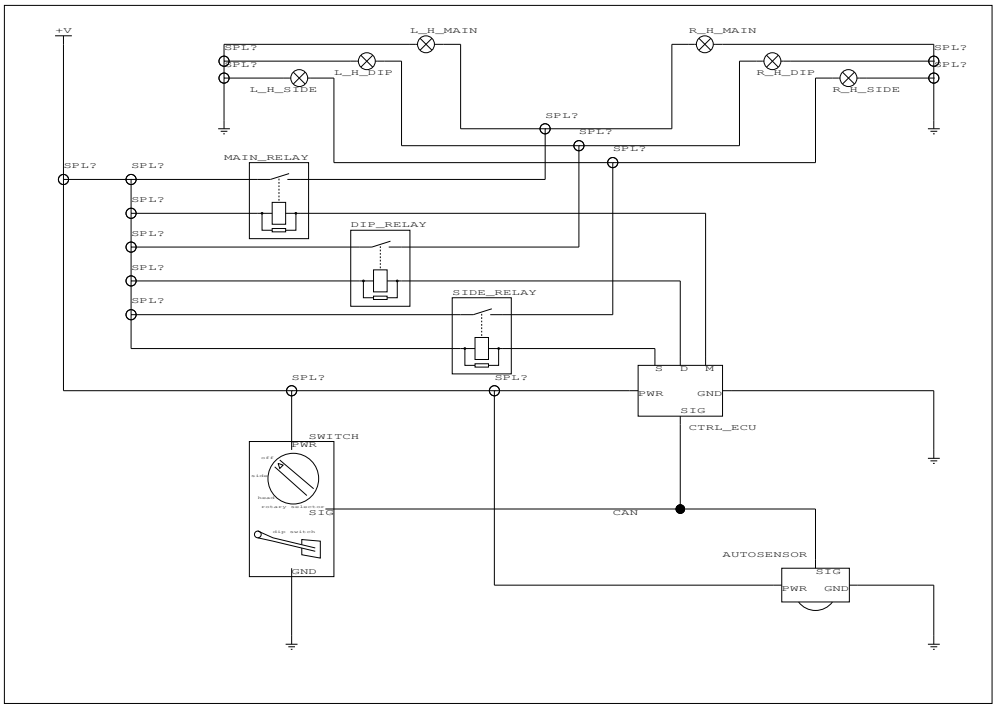

Figure 2: The lighting circuit, without electrical connections between the switch and ECU

electrical simulation. This means that the connections between the components are reintroduced, but their failures will need to be modelled specifically, rather than relying on the domain level rules. It might be hoped that avoiding the intermediate electrical simulation would solve the timing problems associated with the original version of the circuit. This was actually found not to work, and this version of the circuit behaved in exactly the same way as the original version as far as collisions were concerned.

It seems that the cause of the problem with the original circuit is that the state simulation and electrical simulation do not share a common notion of time, so get "out of step" with each other. This seems to be a problem whenever the interaction between individual components in a system is shared between behavioural level (state chart) simulation and domain simulation. Is it feasible to introduce a common notion of time for both simulators? CIRQ, of course, has no notion of time, the electrical events that might change the state of the circuit are implicitly assumed to be instantaneous and not specifically specified as such, which would allow some coordination between the circuit and state simulators.

2.3 Message priorities

In CAN, if two messages collide, the higher priority message is transmitted and the lower priority message transmitter will yield to it. The AutoSteve bus support allows setting of priorities to model this. In CAN, the message's priority is given by its message id, which must be unique. The data bus support includes a message data type having an id and the content. The priority could be set to the message's id (it has to be done specifically). In version 2.6, if the message id was used as its priority this would have resulted in incorrect behaviour as in CAN the lower the message id the higher the priority and in AutoSteve, the higher numbers had higher priority. This has been changed in version 3.1, however, so the message id can now be used.

Naturally, as there was no collision, setting message priorities with the circuit in figure 1 made no difference to its behaviour. Setting the priorities for the circuit in figure 2 means that no collision is detected, and the higher priority message is transmitted.

In AutoSteve, the default behaviour if two messages collide is for the lower priority one simply to be lost, while in CAN the message will be re-transmitted once the other message has cleared the network. (It might, of course, lose arbitration again!). Some time has been spent attempting to model this behaviour with only limited success. To make the outputs more distinct, a more elaborate version of the circuit was used, shown in figure 3. The aim in simulating this circuit was to try to get the behavioural models of the autosensor and the brake pedal to retransmit when the message on the bus is not the one sent by that node. This has been found to be possible, but has not yet resulted in the system as a whole working as expected. At present it is unclear whether the problems are due to my incorrect use of the facilities or whether they are genuine problems. I shall briefly list the findings found here, a more detailed description can be provided, but might be better done in a demonstration rather than in writing.

One problem is that it was impossible to use concurrency groups for the two behaviours of the rear lamp control ECU, for the tail lamps and the brake lamps. This was because they need both to read the same message from the CAN — even though one will ignore it, it must read each message if only so that its irrelevance can be established. It was found necessary to model that component's behaviour more in terms of receiving and waiting for messages from the bus rather than its internal state (which lamps it should be driving). The best approach to modelling a component that receives messages seems to be to model the receiving behaviour in one concurrency group and its ECU behaviour(s) in others. The message signal is used to set values of internal variables and these in turn used to trigger transitions in the ECU behaviours. This has been tried and appears to work.

When attempting to get a transmitter to detect that its message has not been received, the major problem appears to be that a component does not seem to receive its own messages. Detecting that a message has lost arbitration by reading the id of the message on the bus works when the component does lose arbitration

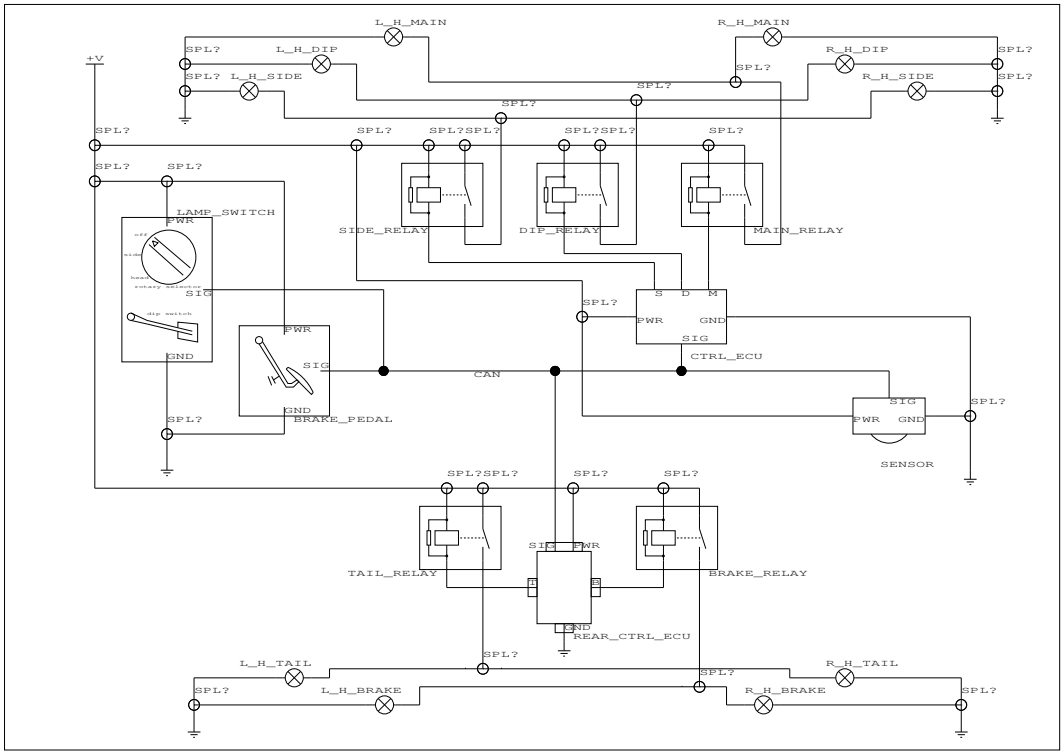

Figure 3: Lighting circuit, adding tail lamps and brake lights

(there is another message on the bus) but the same test cannot be used when there is no collision. This means that the component behaviour freezes in an unexpected state. I have tried using the presence of data on the bus as the test for a collision and this seems to work provided there are only two competing nodes and there is a delay (milliseconds) before testing that there is no data available on the bus. This has not worked in all cases, and seems like a rather hacked work around. It is not clear how this affects modelling of components that win arbitration, as the highest priority component never needs to retransmit this behaviour has not been added that component, as to save time the existing light switch was used and left with highest priority.

2.4 Message latency

The data bus support allows the user to set the message latency in building the models. As in CAN this is undetermined (it depends on the frequency of collisions and how arbitration treats the specific message) it is unclear how useful this is for CAN and other CSMA/CD protocols.

It was suggested that its use might help with the collision problem discussed

in Section 2.2, but it was found not, the problem being caused by the intermediate electrical simulation, as noted there. However, setting the message latency to 100mS for both messages was found to affect the behaviour of the system in figure 1. Instead of only the first message (that from the autosensor)being read, both were read and acted on in sequence, so the simulation ended with the sidelights on, the dipped headlights having been on for several earlier steps in the simulation. This is broadly correct for the situation where the autosensor message happens to get transmitted first but as both messages were written before either was read, there should have been a collision.

It is perhaps worth noting that if two input events occur simultaneously, there are three possible results. Either message might be transmitted first, or they might collide (which with CAN will have the same effect as the higher priority message being transmitted first). This raises the question of whether each of these possibilities should be simulated.

2.5 Message transmission types

According to (Ludwig and Palm 2002), there are four transmission types used for CAN messages. These are:-

Fixed periodic Message is sent periodically, whatever the values are.

Event Message is sent if at least one signal has changed since last transmission of the message and the specified minimum delay (since sending the previous message) has passed. If the minimum delay has not passed, transmission will wait until it has.

Fixed periodic and event Message will be sent periodically (as above) and also sent when at least one signal has changed, after a minimum delay.

Enabled periodic Message is sent periodically provided at least one of the signals differs from its default value. When all signals return to their default values, the message is sent one final time.

It is worth noting that a typical CAN message might include several data. Each individual datum is what are meant by a "signal". For example in the C214 messages, there is a message sent by the transmission control unit that includes the readings for the transmission torque loss, the turbine speed, the drive shaft speed and the transmission ratio, each of these is a "signal". So one message typically includes several signals.

It will be appreciated that modelling components using state charts is best suited to modelling messages of the "event" transmission type. It was noted above (in the introduction to this section) that this has been used for modelling the output of the autosensor component and that this is unlikely to be a correct model of a sensor's behaviour. The question is whether it can be modelled more accurately and whether this is worth modelling.

It would, of course, be perfectly feasible to have a state chart sending a message periodically, by simply adding an ongoing cycle of "fire after" events that send the message. This will result in the simulation being unable to run to a conclusion as the cycle is endless and is arguably rather meaningless if the simulation is being run on an individual system. There are two main arguments behind that statement. The first is that the modelling of network messages in real time (or a scale model thereof) is irrelevant since the rest of the simulation is not run with any consistent notion of time. In the case of the autosensor in the example circuit, the sensor will be sending out a periodic stream of messages until the whole system is closed down. Eventually, one of these messages will include a value that crosses the threshold. This is incapable of being correctly modelled if the system runs to a steady state and then waits for the next change of input. If the message is sent at a fixed period, then if the simulation runs to a steady state and "waits" for the next input event, how does the simulation interrupt the series of transmissions, and add the likely delay in sending the message? If this is done, does it need to test for all possible delays, in other words to assume that it resumes its simulation at any point in the periodic interval between messages? The other argument is that the main advantage of modelling periodic message passing is to use that as a way of modelling the load on the network as a whole. This is obviously not going to be interesting at a system level, unless all nodes on the network are included in the system concerned.

One related difficulty with the example system is that the sensor "knows" when the light level changes relative to the threshold. Realistically, of course, the sensor should send a periodic numerical value and leave it to the control ECU(s) to decide its relation to the threshold value and so whether to change the state of the lights.

Having the autosensor in the circuit in figure 1 send messages periodically has been tried, but it appears that the simulation does fail to reach a conclusion. This is to be expected, as a steady state is never reached, of course. It would be necessary to have the simulator recognise that a cycle had been reached and stopping on doing so. This is necessary for correct modelling of CAN based sensor data messages, as these seem generally to be transmitted on a periodic basis.

The idea that modelling periodic messages needs a consistent notion of time to be useful does seem to have some common ground with the problem identified in Section 2.2. There is also some common ground with the idea that some notion of timing might be necessary for modelling input events whose effects depend on their temporal relationship to other input events, such as the quick unbuckling of the driver's belt in the "belt minder" system being used to temporarily disengage the system.

There seems no particular difficulty in modelling either the "fixed periodic and event" or the "enabled periodic" transmission types using state charts other than the problems outlined above for modelling periodic messages in general.

It is perhaps worth noting briefly that there are two distinct classes of sensor reading. One is those that are unaffected by the system state, as in the case of the lighting circuit example, where the fact that the headlights have been turned on will not affect the sensor (or they will just get turned off again!). The other case is

where the sensor does complete a feedback loop, such as a temperature sensor in a space heating system. In this case, of course, the expected behaviour of the system is that the state will eventually change with no further user input. For example, the heater will switch off once the desired temperature has been reached. This has been discussed in (Bell 2002) and is not considered further here, it having been decided that this line of investigation is beyond the scope of the project.

It is suggested that for simulation at a subsystem level, for FMEA, it is likely to be useful to model messages as being event triggered, in other words to fail to model those that result in no change, at least in cases where feedback is not a factor. This is so if we can model the effects of a message delay in some other way (discussed below) and if we suppose that the possible delay in the system learning of a change of value that results from the normal operation of a periodic message is acceptable. This is a valid supposition for FMEA where the aim is to find the effects of failures, but it would not be a safe assumption for design verification.

2.6 Conclusions

It seems worth briefly summarising the findings of this work. There seem to be areas where the current AutoSteve bus support does not seem quite sufficient for correct modelling of systems that make use of CAN, though it should be noted that some, at least, of these problem areas might be because I have not found the best approach to modelling such systems. These areas will be noted in these concluding notes. The main findings and problem areas seem to be:-

- The use of the message data type seems to work well, and its id can be used both to sort out relevant and irrelevant messages (so allowing "full CAN" behaviour to be modelled, if the CAN terminal is modelled as a separate component) and also to set the priority of messages in handling collisions. It is worth pointing out that I have not tried modelling complex CAN messages that include several data (signals) but I see no reason to suppose that this should not work, using an array of integers.

- This does lead onto the difficulty experienced modelling CAN's collision handling behaviour. It is fair to summarise this by noting that a good deal of time has been spent trying to model the the retransmission of messages with only limited success. However, it it does seem possible that my result could be improved upon.

- The fact that the simulator runs to a steady state and that this means that periodic (and other) message types cannot be adequately modelled does seem problematic, partly because of the difficulty noted earlier, that it becomes desirable to associate behaviour with the wrong component (the sensor only sending a message once the threshold has been crossed) and also if network loading was to be modelled, as this will depend on the modelling of periodic messages. To model this correctly, the simulator needs to be able to recognise that the behaviour is cyclical and stop the simulation gracefully. While

this seems quite feasible, it is appreciated that it is not likely to be a trivial alteration.

- The modelling of collisions raises one or two questions, though these are partly sidestepped by the use of message priorities. One is the idea that two simultaneous input events need not result in a collision (either one might result in its message being sent first or they might both result in changes to different signals in the same message). Should all possibilities be modelled here? Are there cases where the ordering might affect the final result? Obviously, if the two inputs make different changes to the same message, this implies a component whose task it is to construct the message and it should be possible to model this with the rider that the message cannot be sent periodically. The other difficulty is the loss of consistent timing when there is an intermediate electrical simulation. The use of priorities does not help here and there seems to be no answer simpler than the idea that the individual simulators need a common notion of time, so they models stay synchronised. As an aside here, it is worth noting that some sort of global time might also allow the timing of input events to be related, as was found necessary in modelling the temporary deactivation of the belt minder system.

In conclusion, it does appear that the bus support can be used for simple models of simple CAN system, but the problems noted herein seem to prevent correct modelling of more complex (realistic?) cases. There is room for more work in evaluating some of these areas (particularly modelling message retransmission) and it might be suggested that the problems with global time and simulation to a steady state lead to more research questions that should be followed up.

3 Modelling network failures

As suggested in the introduction, it seems worth adding some discussion on modelling of network failures to this report. This material is more speculative in nature, at present no attempt has been made to to model network failures, though it seems likely that some, at least, could be modelled.

In AutoSteve, the data bus is represented using a network (or wires?). A network appears to have no failure modes associated with it, but they could, of course, be added. If wires can be and are used, the failure modes will be inappropriate for modelling network failures, being intended for electrical ones.

In (Thomas 2001) there are six generic network failures listed. These are:-

- Unable to transmit data.
- Unable to receive data.
- Data transmitted late.
- Data received late.

- "Bad" data transmitted.
- "Bad" data received.

These can be paired off, in that the effect of data being "lost" or late or corrupt will be similar whether the cause is failure in transmission or reception. As the causes are different, the "occurrence" value will differ. It could be argued that there are actually nine failures, in threes, where the failure is caused by the sending node, the network itself or the receiver. This idea is suggested by the fact that it is unclear whether a network failure results in transmission or reception failing in (Thomas 2001). In practice, the probability of data being corrupted by the network and that corruption being undetected are vanishingly low, but this does not stop wrong data (from a faulty sensor, perhaps) being accurately transmitted, of course. A detected corruption will result in delay to the message, as it will need to be sent again.

There does not seem to be any great difficulty in modelling non transmission or non reception of messages, especially as the faults that lead to this are likely to be persistent. If a node is to fail to transmit a given message, it is likely that it has stopped transmitting altogether. Similarly, if the CAN stops transmitting, it will cease to transmit any message or if a node stops receiving any message it will stop receiving all messages.

One difficulty with modelling late arrival of a message is that this is not a consistent problem — it is not realistic to model the CAN as delivering all messages late. It is likely that what is wanted in system level FMEA is to establish the consequences of any individual message being delayed. This is, of course, rather different from attaching the failure to the component itself, but presumably it can be attached to the component, the message being regarded as an "instance" of the network's operation, with that failure attached to the specific message.

Some complication attaches here, of course. If a message is delayed because the CAN is overloaded, then all messages (at least all messages of lower priority) will also be affected while if it is because of some fault in the sender, then no other message will be affected. This raises the possibility of the ordering of messages being changed. Do both possibilities need modelling?

There are also two possible consequences of delay to a message. It might actually affect the behaviour (which there should be no conceptual difficulty in modelling). One obvious case here is the possibility that ordering of messages is changed, though this is, of course, next to impossible if a network (as opposed to transmitting node) failure means a lower priority message being transmitted before rather than after a higher priority one. The other case is where the effect is simply that a system function is achieved late. The late achievement of functions has been considered in (Joseph 2003) and (Bell 2003a) and is not considered further here. It is also the case that it might be necessary to simulate systems whose functionality depends on relatively complex behaviour (the belt minder system is an example even though it does not use a data bus) and this might lead to further problems with the existing functional modelling language. These problems seem more apparent for tasks other than FMEA, where is is more need to link the system functions output with the

system input state, such as in SCA or design verification. This is discussed in (Bell 2003b).

The idea of attaching the failure to the message rather than the network seems a possible approach to modelling corrupt data. One possible difficulty here is that corruption might conceivably result in a message that can normally be ignored having an effect. Should we attempt to model the case where the message id is corrupted, so the receiver acts upon the wrong message? I suggest this is so unlikely that it can be ignored, especially as modelling all possible messages (so their content can be used) seems unduly cumbersome. This leads on to another difficulty of modelling corrupt messages — the effect is likely to depend on the nature of the corruption, to which part of the message (id or signal) the actual value transmitted. As these are typically numerical, there is a range of wrong values that might be transmitted. Does each possible corrupt value need modelling? It is suggested that this difficulty, combined with the improbability of corrupt transmission being undetected means that this failure need not be modelled (except in terms of delay to the message). In CAN if a node persists in sending corrupt messages it will eventually be forced to stop transmitting altogether. This will, of course, be modelled by non-transmission of the appropriate messages. As in a given simulation, all that is required is for those messages to be modelled, this can safely be considered a case of non transmission of data.

3.1 Summary

It seems worth closing with a summary of questions or areas that might need consideration.

- Changes necessary to model late achievement of function. An approach to this has been suggested in (Joseph 2003), and a similar approach in (Bell 2003c).
- Method of modelling time appropriate for modelling message delays. The order of magnitude time is arguably of limited use here as the intervals between the orders of magnitude are so large.
- Possibility of attaching to failure to messages rather than network, so individual messages can be modelled as being late.

4 Conclusion

This report has described findings of attempting to model simple CAN based systems using the AutoSteve bus support and has introduced areas that need consideration if network failures are to be modelled. It is accepted that some of the problems identified might be solved by more knowledgeable application of the facilities provided, but the investigation carried out so far has shown areas where further research might be valuable, and the report also suggests ways in which network failures might be modelled. These include:-

- The need for global time, so different parts of the simulation stay synchronised correctly.
- The need for the simulator to recognise that it has reached a cyclical behaviour and so can stop.
- The possible need to capture late achievement of system functions, resulting from late reception of messages.
- The need to attach failures to individual messages, so as to capture the intermittent nature of that failure mode, the need to establish the consequences of a specific message being delayed. There is perhaps some interest in the relationship between this area and the modelling of periodic messages.

References

Bell, J. (2001a). Comparison of protocols used in the automotive industry. SoftFMEA document ref. SD/BCG/PR/GEN/03.

Bell, J. (2001b). Notes on CANbus. SoftFMEA document ref. SD/BCG/PR/CAN/01.

Bell, J. (2001c). Other automotive industry protocols. SoftFMEA document ref. SD/BCG/PR/GEN/01.

Bell, J. (2002). The heater circuit - matters arising. SoftFMEA internal report.

Bell, J. (2003a). Comments on "temporal functions". SoftFMEA memorandum.

Bell, J. (2003b). Function and complex behaviour. SoftFMEA document ref. SD/TR/FR/03.

Bell, J. (2003c). Temporal aspects of functional modelling for design analysis. SoftFMEA internal report.

Joseph, R. (2003). Temporal functions. FirstEarth report, draft version 0.2.

Ludwig, C. and D. Palm (2002). 2003 C214 CAN RO5 CAN infosheet. Ford confidential memo.

Snooke, N. A. and C. J. Price (2000). SoftFMEA — automated safety analysis of automotive embedded systems. Proposal submitted to EPSRC Critical Systems Programme.

Thomas, T. J. (2001). *Generic Network FMEA*. Ford Motor Company.