Abstracting Automotive System Models from Component-based Simulation with Multi Level Behaviour

Snooke, Neal; Bell, Jonathan

Publication date:
2002

Citation for published version (APA):
Snooke, N., & Bell, J. (2002). *Abstracting Automotive System Models from Component-based Simulation with Multi Level Behaviour*. 151-160. http://hdl.handle.net/2160/68

Abstracting Automotive System Models from Component-based Simulation with Multi Level Behaviour

Neal Snooke, Jonathan Bell
Department of Computer Science, University of Wales
Aberystwyth, Ceredigion, SY23 3DB
email: nns@aber.ac.uk

Abstract

Recent work in Model Based Reasoning has resulted in the development of automated tools to perform Failure Mode Effects Analysis (FMEA) and Sneak Circuit Analysis (SCA). These tools work at the component level for individual systems or subsystems. Analysis of multiple systems is becoming necessary because of the increase in system and subsystem interactions resulting from Electronic Control Unit networked system architectures.

This paper considers the automated construction of black box system or subsystem models from envisionments produced by a mixture of qualitative electrical and finite state machine simulations. These models can then be used as part of a larger simulation to promote reuse of simulation results during the repetitive automated FMEA task and hence improve simulation efficiency when multiple systems are involved.

The approach also has the potential to allow Sneak Circuit Analysis (Price, Snooke, & Landry 1996; Price & Snooke 1999) to be extended to include systems with internal memory. It provides the basis of a useful visualization tool for complex system behaviour.

Keywords: Qualitative simulation; FMEA; State based model; System abstraction.

Introduction

FMEA is a repetitive design analysis task involving simulation for every potential component fault. The aim of this work is to allow the successfully deployed AutoSteve FMEA tool (Price *et al.* 1997; FirstEarth 2001) to work with larger groups of systems typically linked by networks such as the CAN[1] bus. The qualitative electrical representation used in the Autosteve tool allows analysis to be performed early in the design process for the majority of automotive systems and faults. Other work is currently in progress to provide targeted quantitative analysis for situations where the qualitative approach does not provide enough detail, and also late in the design process when numerical values are known.

[1]Controller Area Network

Automotive industry practice generally requires only single fault scenarios to be considered, allowing an abstracted system model to replace the component level simulation for the systems of a multi-system analysis that *do not* include a fault. This can provide big savings in simulation effort during an entire whole vehicle FMEA without losing the effects of a fault on other systems. The abstracted system models must adhere to the 'no function in structure' principle since faults in neighbouring systems often cause abnormal external environments to an non-faulty system. Abstracted models computed automatically from an attainable envisionment (Kuipers 1994) of each system have the advantage that they will be an accurate description of system behaviour in all circumstances and do not involve engineer time to construct.

This paper describes a technique for deriving a simple unambiguous system level model faithful to the behaviour at the system boundary. The resulting model is far simpler than the envisionment because the envisionment includes the state of every component of a system and represents also the component interaction produced by the electrical simulator. The majority of the model variables (envisionment states representing variable value combinations) are not significant to the overall system level behaviour, and in general, single component model variables do not efficiently represent system level states. The envisionment process usually produces a large, deterministic, but reducible FSM. Work on FSM abstraction generally assumes no exact smaller equivalent exists and then proceeds to lump the inputs, outputs, and states to produce a nondeterministic or probabilistic FSM (Oikonomou 1996; 2001). We only lump the states in this work, to generate a FSM that is equivalent at the system interface to the envisionment.

The simpler model has several potential benefits. It can provide an engineer with a more comprehensible model of the system behaviour. Anomalous behaviour is observed as unexpected system states, event transitions with unexpected conditions, or an asymmetric set of events (closing the switch causes a state change but opening it again does not return the system to the same state for example). In the future the sys-

tem level internal state information will be required to perform SCA, fault signature analysis (Spangler 1999), and automated scenario generation for FMEA. An important pragmatic consideration is that no additional model building effort is required from the engineer.

System Modelling Overview

A detailed account of the models and simulation (Qualitative Circuit Analysis Tool known as QCAT) are provided in (Snooke 1999; Price *et al.* 1997; Lee & Ormsby 1993). In this section we provide an outline of the salient features.

The system models are comprised of a netlist generated from system schematics together with a library of reusable component models. Each component model contains an electrical structure defined by a network of qualitative resistances and optionally, a behaviour defined as a Finite State Machine (FSM) (Harel & Politi 1998). The FSM events (state changes) are triggered by the results of electrical simulation or changes to the environment (inputs/outputs) of the component. The FSM also controls the values of the resistive circuit elements triggering further electrical simulation. Simulation takes the form of a sequence of DC electrical analysis steps using qualitative signs driven by both external events and higher level component FSM behaviours.

Components and systems simulated using QCAT may have static or transient internal states (memory) typically attributable to the following sources in automotive systems:

Non electrical components An example is the mechanical toggle switch. The toggle button contacts change position for each complete press and release cycle.

Electronic Control Units ECU components are microprocessor based modules that may contain significant internal states. Software operation is modeled at a high level by FSM descriptions.

Feedback Emergent state can be produced in systems that include feedback circuits even though the individual components have no internal state. An obvious example is a bistable constructed from 2 NAND gates.

Time based transitions Component models can include order of magnitude time constraints applied to events. This allows resolution of some of the potential qualitative ambiguity caused by multiple events. All events in each time period are run prior to those in the next longest time period regardless of how many there are. A typical automotive sequence of timeslots represents: instantaneous (electrical propagation); μS (ECU operations); mS (relay switching); S (user interaction); hour (battery discharge). These delays are equivalent to short term memory in the system.

The interactions with a system are provided by the set of variables that define its environment. This system environment is a subset of the *component* inputs for example the position of a switch. It is a typical system characteristic to have relatively few inputs and outputs compared to the number of internal components thus ensuring that abstraction is possible.

Envisionment

The attainable envisionment (Forbus 1990; Kuipers 1994) represents all states of the system reachable by exercising any sequence of external interactions from the default state. The envisionment is generated by a depth first search, terminating each branch when a previously encountered *envisionment state* is found.

Each state of the envisionment represents a specific set of the possible values of the model variables. For our models, envisionment state is a composition of:

- all component FSM model states;
- all resistive element values in the netlist;
- all input and output variables included in component models.

The System Perspective

By definition the components of a *system* will work together concurrently or in sequence to produce functions of the system that provide coherent effects in its environment (Chandrasekaran & Josephson 1996). This high degree of dependency in the operation of the individual components ensures that relatively few of the possible component state combinations will ever be realized during non-faulty operation of the system. The generated model must include only internal state significant at the *system level*. It is not generally possible to include this state directly from the components because:

- system states comprise composite component state;
- component behaviours are partially used within the system;
- the connection topology introduces causal cycles (feedback);
- component internal states are reflected at the system interface.

An example will clarify the final point. Consider figure 1 (circuit A). It is possible to deduce that if the lamp is **off** and the switch becomes **closed** then after some time (**mS**) the lamp will change to the **on** state. If the switch becomes **open** in the intervening period the state does not change (**mS** event condition is no longer satisfied). The system is therefore described by the 2 state diagram. Circuit B uses the same components and system interface however its behaviour is different. When the lamp is **off** and the switch becomes **closed** it is not possible to determine when the lamp illuminates (immediately or after a **mS**). This depends upon how recently off state was achieved and hence the internal state of the relay contacts. The relay contributes no net

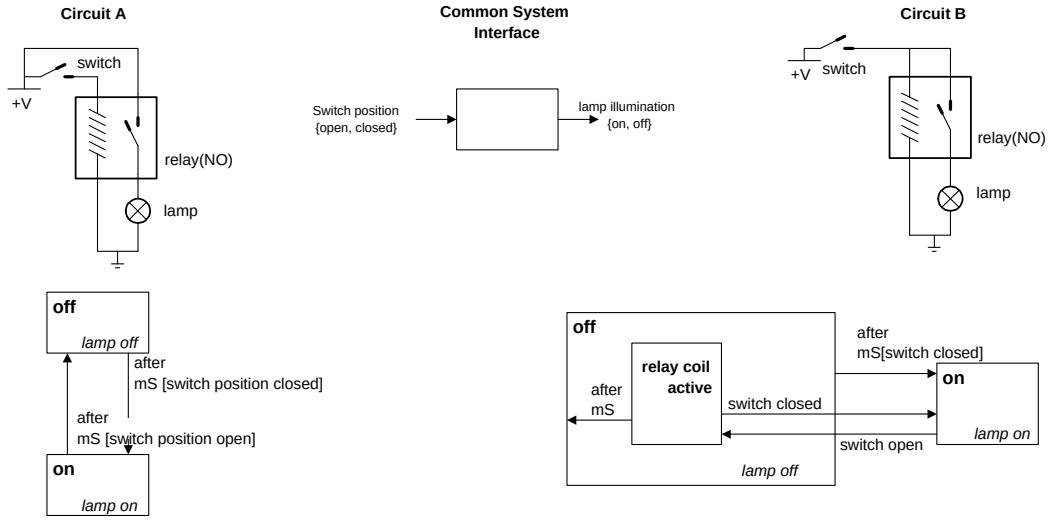

Figure 1: Reflecting component state at the system interface

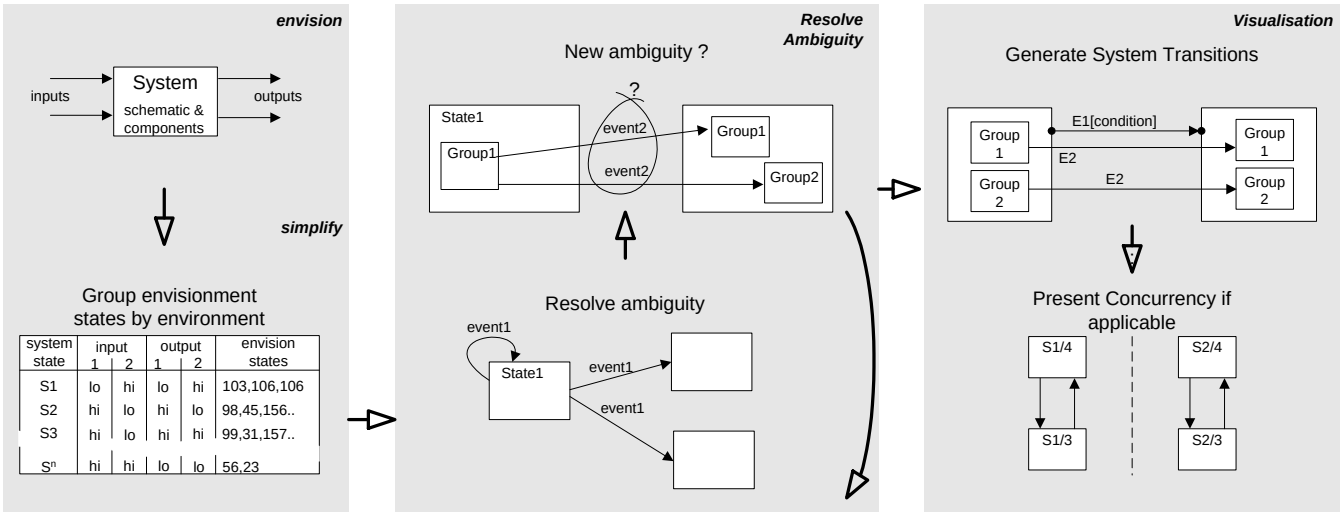

Figure 2: Simplification overview

internal state to the first system, but adds a transient state to the second.

The strategy used to generate the system model is to produce an over simplified (ambiguous) black box model based on the system environment and then derive component based details until the model no longer contains ambiguity. The major steps are depicted in figure 2 as follows:

Simplify Combine attainable envisionment states using system boundary.

Resolve Ambiguity The introduction of new state in this step may introduce new ambiguity resulting in an iterative algorithm. This leads to the extraction of system level internal state.

Visualise Extract system level events between detected system states and orthogonalize the representation.

Envisionment Abstraction

The initial set of system states are obtained by grouping the envisionment states by the values at the system interface.

In the subsequent description transitions between envisionment states are referred to as *envisionment transitions* and transitions between system states are *system transitions*. *System states* are sets of related envisionment states and therefore system transitions represent one or more envisionment transitions. The only transitions that can be significant at the system level represent changes to the *system environment*. Each unique set of changes to the system environment is a *system event*. Several system transitions may represent the same system event. Time is also considered as a feature of the environment of the system since it is externally imposed.

Identification of Insignificant System States

Envisionment transitions leading to new states that differ only by the environment change specified by the associated event are not interesting. Such states may be combined since the environment of the system can be used to distinguish multiple transitions of an event by adding additional conditions. Figure 3 illustrates the combination of states **es1** and **es2** into **S1**. The resultant system state **S1** represents multiple values (or component states) for some system variables. In the figure **ip** can have the value **A** or **B** in the system state **S1**.

Using a simple automotive example, it is clear that closing the ignition switch in the system 'off' state will cause a different transition dependent upon the existing position of the lighting switches. It is not desirable however for the system model to include these component states (switch contact resistances etc) when the ignition switch (and system) is off.

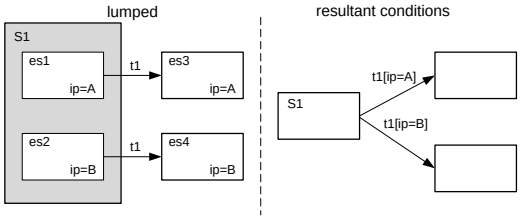

Figure 3: Insignificant states

Implicit Events

During the envisionment, transitions are not produced for external events that cause no state change. This provides a complete representation since all possible events are simulated from every state. An *implicit* transition loops back to its originating state for all ineffectual events. Time based events are the most common example. They notionally occur for each possible qualitative timeslot but only those that cause a state change will be made explicit by the envisionment.

Once envisionment states are combined, implicit events are a source of ambiguity. Figure 4 illustrates this point. The implicit event **t1** from **es3** must be considered since otherwise it is not clear what happens in **S1** when **t1** occurs.

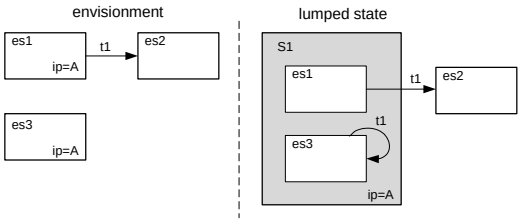

Figure 4: Implicit events

For this reason all envisionment transitions that represent system state changes must be differentiated from all other envisionment states in the group by considering implicit events where necessary.

Transition Ambiguity

Ambiguous transitions are those that create alternative behaviours in the model. In this work we have divided into two categories; ambiguous and semi-ambiguous. Ambiguous transition pairs represent the same event from the same source leading to different destinations (system states). Semi-ambiguous transitions can be distinguished by a difference in the system environment Ambiguous transitions occur because of internal system state that is missing from the representation and will be considered in the next section.

It has been assumed that different system events cannot be ambiguous. This assumption is true if system events represent single (unique) input changes. Where multiple simultaneous changes are used, an event (E1)

may represent a subset of the changes associated with another event (E2). E1 is therefore ambiguous with E2. The uniqueness properties of all events used in an envisionment are determined at the outset of simplification and are taken into account when subsequent transition ambiguity is checked.

Internal State and Memory

System memory will lead to ambiguous events in the model. There is guaranteed to be at least one distinguishing feature (ie component variable or state difference) between any two events in a deterministic system since no two envisionment states can be identical, and any given event applied to an envisionment state must always cause the same effect.

The system internal state may represent a single feature or combination of features belonging to the component models, and is represented using lumped *state groups* for each state of the system environment.

Ambiguous transitions are resolved by finding a set of distinguishing factors. System states with ambiguous transitions are then split into state groups using the values of these factors. Factor sets are found using a best first truncated search as follows:

1. The minimum solution length is set to the number of distinguishing factors found.
2. The (remaining) factors are ranked by the number of ambiguous transition pairs they can distinguish.
3. If any factors exist that are the sole distinguishing factor for a transition they are all selected.
4. else select each factor in turn by rank order.
5. For each selected factor(s)
6. Include factor in the possible solution. Remove transitions it can distinguish from list.
7. If the transition list is empty a solution has been found - set the minimum solution length
8. elseif the partial solution is shorter than the minimum solution length recurse from 2 using the reduced transition list

The search strategy rapidly finds a solution using the fewest number of factors allowing the majority of the search tree to be discarded at an early stage. All of the solutions using the minimum number of factors are found.

The minimum number of factors on a system wide basis are chosen by selecting one of the solutions for each system state, new state groups are generated, and the states forming the sources of ambiguous transitions are allocated to the new state groups. Notice that envisionment states are allocated to groups only to resolve ambiguity, and not meerly because they contain the factor values associated with the group. This ensures that groups represent 'actual system memory *use*' rather than 'component memory *capability*' i.e. the memory must be significant to the future system behaviour.

For automotive circuits the same few factors are usually responsible for all ambiguity in a system. This is unsurprising because the ambiguity is caused by system memory and in the systems we consider this memory is limited to special ECU or mechanical states.

The introduction of state subgroups might lead to additional ambiguities between envisionment transitions. Repeated application of the ambiguity solver is required with each iteration relumping the envisionment states. Consider the situation in figure 5. During the generation of system states the **e2** events all have destinations internal to **S1** and hence are not ambiguous. The state group is introduced to resolve the **e1** ambiguity. Now the **e2** event from the state group is ambiguous with the implicit **e2** event.

The ambiguity resolution process is guaranteed to terminate since *additional* ambiguous transitions are considered at each iteration. In the worst case of an irreducible envisionment, all the transitions would be selected and the result would simply be the envisionment itself. In the worst case the minimum number of factors that could be included at each iteration is one (deterministic envisionment) and therefore the maximum number of iterations possible is simply the number of factors in the system. Empirically we note that a solution is normally found in a very small number of iterations.

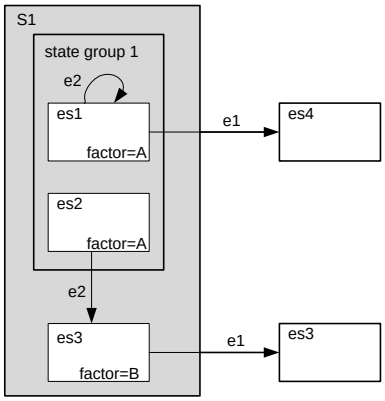

Figure 5: Introduced ambiguity

Semi-ambiguous events

Appropriate environment conditions can be added to remove semi-ambiguous transitions. These additional conditions do not represent any additional changes to the device (input) environment, they are merely checks to determine behaviour based on the environment of the state prior to the event occuring. Environment conditions are shown in square brackets after events in the final model. Notice that the distinguishing environment value can be one that is responsible for the event. However, this is not a problem since the extra conditions refer to the system environment *before* the event occurred. Consider the **Switch.dip_position:released**

[**Switch.dip_position:pressed**] transitions in figure 9 for an example of this.

There can be several semi-ambiguous transitions from a system state and a number of possible environment factors that might distinguish them. The problem is similar to the state grouping discussed before but with the aim of selecting the minimum number of input variables to use as conditions to distinguish otherwise ambiguous events.

Generating the System Events

System transitions are those connecting system states or state groups. The system transitions are generated from envisionment transitions by:

- Removing those internal to a system state group.
- Removing duplicate transitions.
- Combining multiple transitions for the same event between the same system states.

An Example

This section illustrates the result of applying the method to (part of) an automotive lighting circuit.[2] The schematic in figure 6 controls the sidelights and main lights of a vehicle (ign_sw has been ignored to keep the example analysis compact). There is also a dip (hi/low) beam function. In operation the driver changes the position of the switch causing a signal to appear at the ECU input. The ECU then activates an output connected to one or more of the relay coils. The relay models include a qualitative delay of the order millisecond, after switching the appropriate lamps will illuminate. The ECU behaviour is not shown for reasons of space. Its behaviour simply links inputs to outputs as expected except that for the main and dipped light outputs the sidelights also remain activated. Notice that the switch model in figure 7 provides only one of the sidelights, main lights or dip lights outputs at any time. The reason for this is that in the future the link between the console switch and ECU could be provided by messages on a CAN bus.

Outline of the processing

The example circuit produces an envisionment containing 48 states. Figure 8 shows the result of the envisionment. Each state identifier represents a unique state of the system, prefixed with the event that caused the transition to that state. States in bold font are those that have been encountered previously during the envisionment. There are 7 possible external input events in the system each is given a short identifier as follows:

```
side:     Conditions: [SWITCH.selector_position:side]  Timeslot: Time: inst
off:      Conditions: [SWITCH.selector_position:off]   Timeslot: Time: inst
head:     Conditions: [SWITCH.selector_position:head]  Timeslot: Time: inst
mS:       Conditions: []  Timeslot: Time:mS
released: Conditions: [SWITCH.dip_position:released]  Timeslot: Time: inst
pressed:  Conditions: [SWITCH.dip_position:pressed]   Timeslot: Time: inst
```

[2]The circuit is fictional and the behaviour of the switch and ECU is not intended to be complete

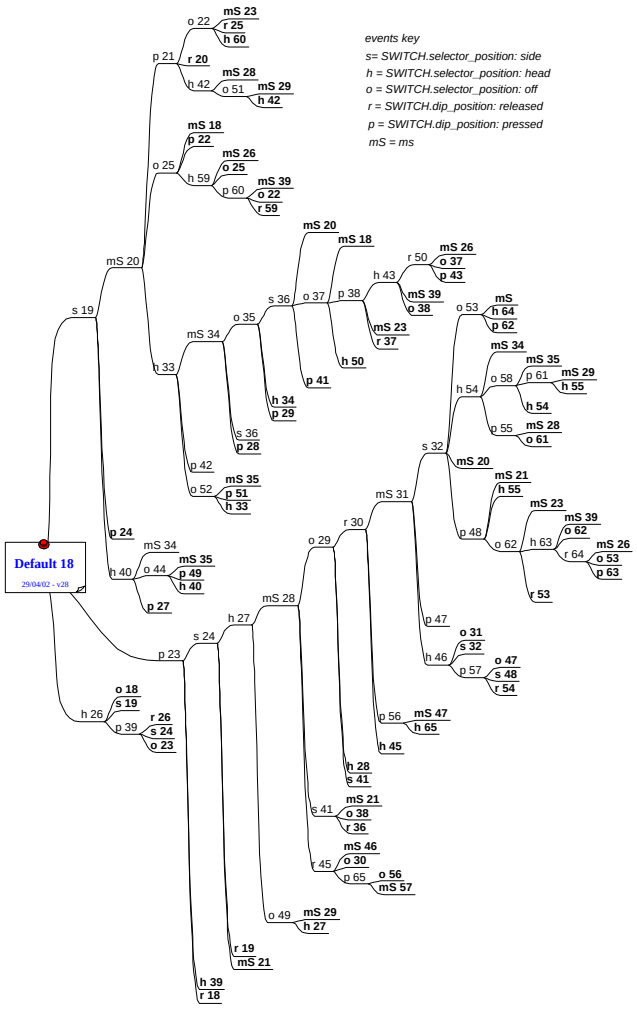

Figure 8: Example system envisionment

The envisionment states are then combined into 4 aggregate states based on the environment, then further combined to take account of insignificant system states as described previously.

agg. state	LEFT _DIP .Light	LEFT _MAIN .Light	LEFT _SIDE .Light	RIGHT _DIP .Light	RIGHT _MAIN .Light	RIGHT _SIDE .Light	SWITCH .dip_pos	SWITCH .selector
1-plus	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	presse releas	off head side
10-plus	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	presse releas	off head side
11-plus	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	presse releas	side head off
14-plus	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	presse releas	side head off

```
Envisionment states for each aggregate state
1-plus  43 65 28 37 35 30 41 56 38 29 50 45 34 36
10-plus 18 44 39 27 24 49 23 40 26 19
11-plus 55 63 57 47 62 61 32 48 54 46 64 53 31 58
14-plus 21 51 22 33 59 20 42 60 52 25
```

The events from each of the aggregate states are checked and any that lead to different aggregate states

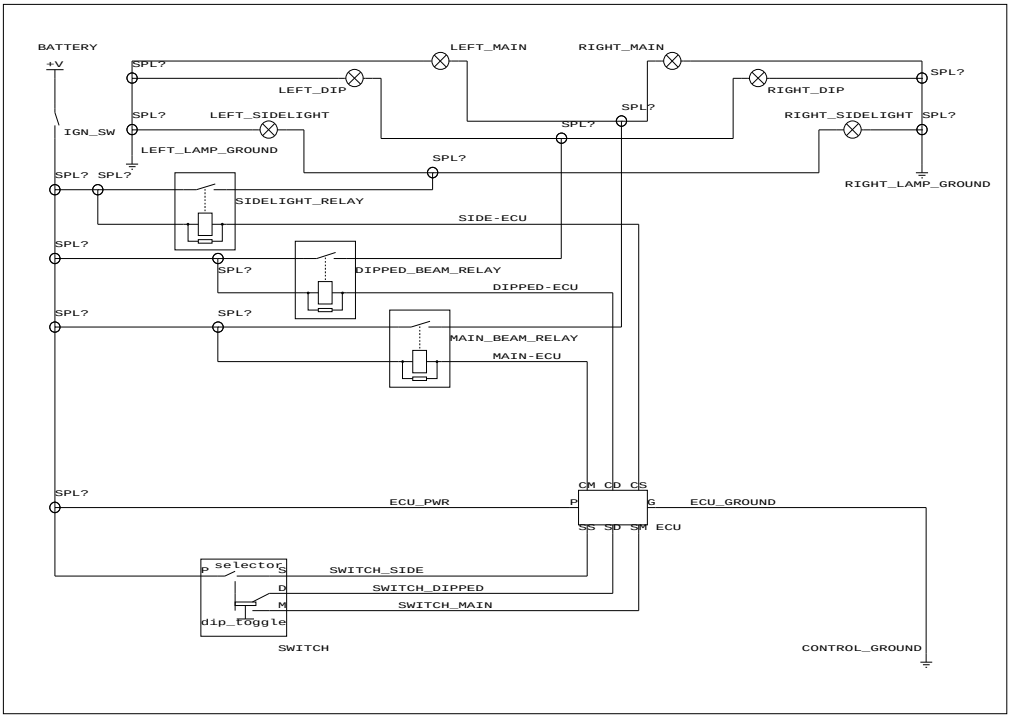

Figure 6: Simple lighting system

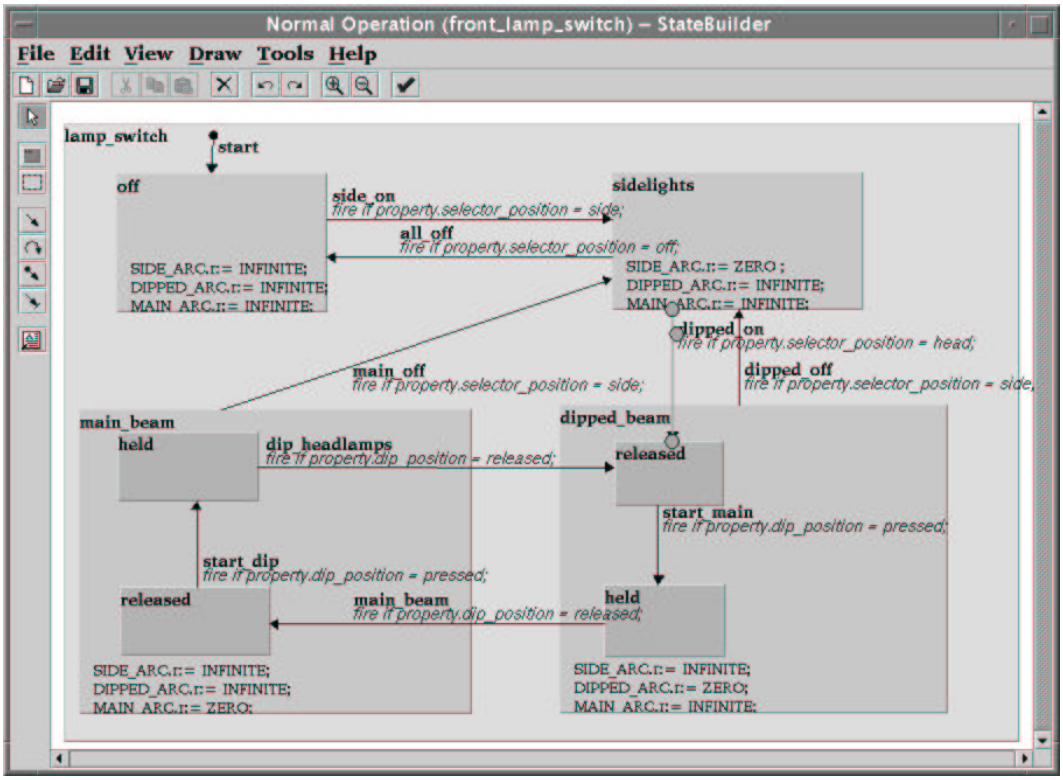

Figure 7: Switch behaviour

are categorised ambiguous or semiambiguous. The example turns out to have 16 ambiguous transitions for each of states **1-plus** and **11-plus**, 4 for state **10-plus**, and 8 for state **14-plus**. For each pair of ambiguous transitions the factors that could be used to distinguish them are considered, and for efficiency, any factors that have identical values for all of the transitions under consideration are treated as a single entity. The most abstract factor is chosen as the representative, ie states in preference to electrical values, and parent states in preference to child states. In the following output for one of the lumped states each column represents a factor. Factors that can distinguish each ambiguous pair of transitions (row) are marked with an `X`. Transitions marked with a single state identifier represent *implicit* transitions a state.

```
AMBIGUOUS TRANSITIONS State: 1-plus
factors are: 1 MAIN_BEAM_RELAY.Power_off 2 SWITCH.off 3 ECU.main_beam
 4 SIDELIGHT_RELAY.Power_on 5 SWITCH.held 6 DIPPED_BEAM_RELAY.Powering_off
 7 SWITCH.dipped_beam 8 SWITCH.released

                        12345678
{43_to_39 65_to_57}     XXXXX
{43_to_39 28}            X XXXX
{65_to_57 43_to_39}     XXXXX
{65_to_57 28}           X X  XX
{37_to_18 35}              X X XXX
{37_to_18 30_to_31}       XXXX   X
{30_to_31 37_to_18}       XXXX   X
{30_to_31 35}           X X  XX
{56_to_47 38_to_23}     XXXXX
{56_to_47 29}           X X  XX
{38_to_23 56_to_47}     XXXXX
{38_to_23 29}            X XXXX
{50_to_26 45_to_46}     XXXX   X
{50_to_26 34}            X X XXX
{45_to_46 50_to_26}     XXXX   X
{45_to_46 34}           X X  XX
```

Each factor is ranked according to its ability to distinguish transition pairs. In this case 4 factors occur 12 times and **MAIN_BEAM_RELAY.Power_off** is chosen first. This leaves only 4 ambiguous factors for the second level of search. **ECU.main_beam** is removed because it cannot distinguish any of the remaining rows and **SWITCH.off** is chosen as the first of the best 4 factors thus providing a 2 factor solution. The search continues truncated at 2 factors. In all 24 two factor solutions are found with no single factor solutions and in total only 9 factor combinations are checked which fail to provide a solution.

The potential solutions for all 4 aggregate states are considered together (using a variation on the above search) to choose one of the sets of factors for each aggregate state using the minimum number of factors in total. 4 possible sets of factors are derived:

```
SWITCH.dipped_beam ECU.main_beam
SIDELIGHT_RELAY.Power_on SWITCH.dipped_beam
SWITCH.off ECU.main_beam
SWITCH.off SWITCH.dipped_beam
```

Each of these will remove the identified ambiguity once the aggregate states are lumped according to these factors. The last factor sets is chosen because it involves the minimum number of different components (only the switch). Choosing any set appears to produce an equivalent model of the same complexity. The aggregate states are now subdivided according to the factor set chosen:

```
Selected factors: [SWITCH.off, SWITCH.dipped_beam]

Grouping State: 1-plus
Using following factors for grouping: [SWITCH.dipped_beam, SWITCH.off]
Factor set: SWITCH.off; 43 37 38 50
Factor set: SWITCH.dipped_beam; 28 35 29 34

Grouping State: 10-plus
Using following factors for grouping: [SWITCH.off]
Factor set: SWITCH.off;  18 39 23 26

Grouping State: 11-plus
Using following factors for grouping: [SWITCH.off, SWITCH.dipped_beam]
Factor set: SWITCH.dipped_beam; 55 61 54 58
Factor set: SWITCH.off; 63 62 64 53

Grouping State: 14-plus
Using following factors for grouping: [SWITCH.off]
Factor set: SWITCH.off; 22 59 60 25
```

The ambiguity check is now performed again and no ambiguity is found. If new ambiguity had been introduced the new set of ambiguous transitions are added to the list of ambiguous transition list and the whole process repeated.

Result

The model in figure 9 was generated automatically[3] for the example system. The internal states of the system are shown inside the four externally visible system states. It is clear that the four main states of the system (off, sidelight, main beam and dipped beam) are represented based on the state of the lamps.

The delays introduced by the relays are also clear as states with **mS** exit events (transition occurs after a millisecond order time period). These delays also lead to some less obvious event conditions because the switch may have been moved several times during the relay delay. For example the **10-plus** and **14-plus** state groups include the pair of transitions **off**[**side**] and **Time:mS**[**head or off**]. The timed transition based on the **off** condition could only occur when when the switch input changed to **head** and then directly to **off**. This sequence prevents the system from going back into the off state, a situation also illustrated by several other transitions in states **1-plus** and **11-plus**. Here the transitions containing conditions [**off or head**] seem odd since it is not obvious why **off** should have been included as a condition. Investigation of the switch shows that it is impossible to go directly to head lights without passing through side lights. This constraint does not exist in the environment, with the result that a change in switch position from **head** to **off** will be ignored by the switch model, with the system remaining in the head lights state.

The toggle behaviour of the dipped/main beam switch is illustrated by the **released** [**pressed**] events. These ensure that the switch must be released prior to another change in state and the guard conditions have been generated as part of the process of resolving ambiguous **release** events from the envisionment states represented by the single state shown on the diagram.

An engineer considering the **release** event that includes the condition that the switch should be in the

[3] The layout and label *placement* was edited manually to improve clarity

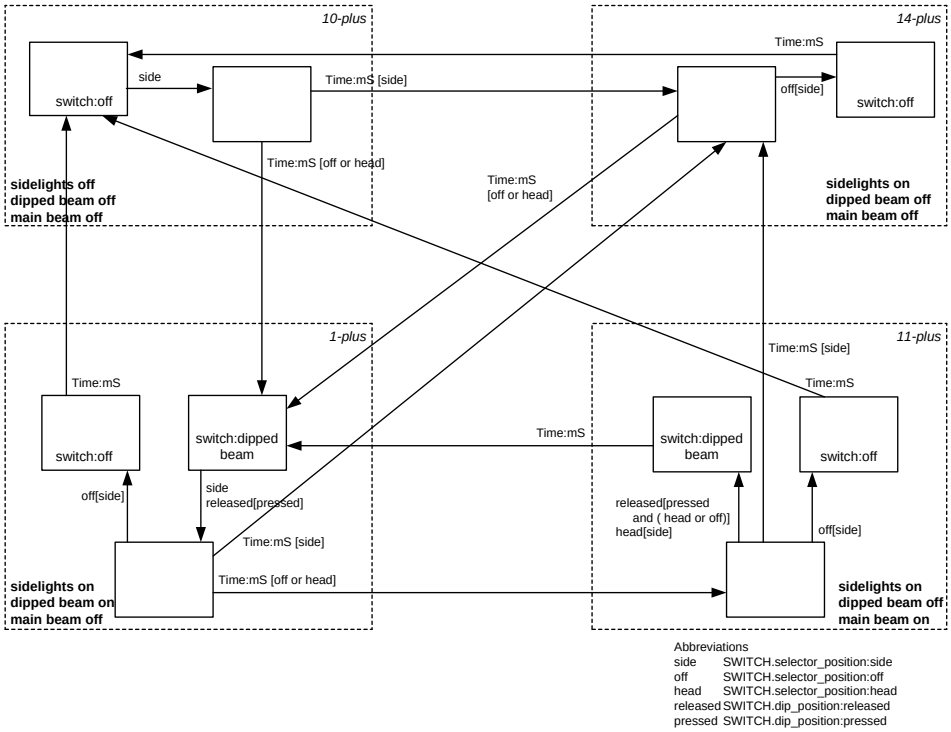

Figure 9: Automatically generated system behaviour

head or **off** positions may well (rightly) question why this state change should occur if the system has the **off** input and hence the problem with the switch will be located.

Discussion

The technique described in this paper has been implemented (in Java) and several automotive system circuits have been successfully analysed. The system state chart produced so far is essentially a 'flat' set of states and the system can be in only one of them at a given time. Therefore for complex systems it is likely there will be many states caused by all the combinations of internal memory values. Many systems have several almost independent behaviours or sub behaviours and this orthogonality can be used to simplify the description using a statechart concurrency notation. The example presented does not contain independent functions and therefore cannot be simplified in this way. If we included the switch and lamp required to implement the 'stop lamp' circuit the resultant system state chart includes twice as many states and an economic representation could be generated by separating the 'stop lamp off' and the 'stop lamp on' states into a totally independent behaviour. The automatic generation of such a representation is currently an area of investigation. In particular it is sometimes the situation that a function is almost but not quite independent and this requires special conditions to be included in the concurrent representation.

It must be *possible* to produce an envisionment of the system. For most automotive system the amount of state is generally limited to mechanical positions, transient state and simple ECU controller memory and these systems respond well to the technique. Circuits with lots of explicit memory, for example shift registers, or counters (hardware or software), will not be amenable to this form of analysis without some form of abstraction of the component state.

We expect to be able to utilize these models in a more efficient version of the AutoSteve FMEA tool aimed at whole vehicle FMEA. This ability is significant because:

- The recent introduction of networks into many vehicles allows groups of systems to exchange messages leading to wider effects for some failures.
- Many of the most dangerous faults are caused by unexpected and subtle interactions between systems designed by different engineering teams.
- The qualitative nature of the simulation allows the above design flaws to be detected early.

The results of this work can be applied to tasks such as SCA and fault signature analysis that require the internal system 'memory states' to be known. The current graphical presentation has scalability issues but we expect that with further development the descriptions should provide a valuable additional abstraction facility to virtual yellow boarding and design verification tools (Price, Snooke, & Ellis 1999; McManus *et al.* 1999). The ability to interactively view specific event categories, conditions, or states, could be provided with such a

tool to enable an engineer to be able to gain an overview of a system behaviour. Many system level behavioural anomalies will present themselves as states or event conditions that the engineer would not expect. Indeed we have already found several examples of existing system schematics where the system model illustrates anomalous system level behaviour in this way.

Acknowledgements

This work has been supported by the UK Engineering and Physical Sciences Research Council (GR/N06052 - Whole Vehicle Whole Life-cycle Electrical Design Analysis), Ford Motor Company, and First Earth Ltd.

References

Chandrasekaran, B., and Josephson, R. 1996. Representing function as effect: assigning functions to objects in context and out. In *AAAI Workshop on modelling and reasoning.*

FirstEarth. 2001. *AutoSteve.* FirstEarth Limited, http://www.firstearth.co.uk/.

Forbus, K. 1990. The qualitative process engine. In Weld, D., and DeKleer, J., eds., *Readings in Qualitative Reasoning about Physical Systems*, 220–235. Morgan Kaufmann.

Harel, D., and Politi, M. 1998. *Modelling Reactive Systems with Statecharts.* McGraw-Hill, 1st edition.

Kuipers, B. 1994. *Qualitative Reasoniong - modeling and simulation with incomplete knowledge.* ISBN 0-262-11190-X: MIT Press.

Lee, M., and Ormsby, A. 1993. Qualitative modelling of the effects of electrical circuit faults. *Artificial Intelligence in Engineering* 8:293–300.

McManus, A. G.; Price, C. J.; Snooke, N.; and Joseph, R. 1999. Design verification of automotive electrical circuits. In *13th International Workshop on Qualitative Reasoning.* Lock Awe.

Oikonomou, K. N. 1996. On a class of optimal abstractions of state machines. *Formal Methods in System Design* 8:195–220.

Oikonomou, K. N. 2001. Abstractions of random state machines. *Formal Methods in System Design* 18:171–207.

Price, C. J., and Snooke, N. A. 1999. Identifying design glitches through automated design analysis. In *Annual Reliability and Maintainability Symposium*, 277–282. IEEE.

Price, C. J.; Pugh, D. R.; Snooke, N. A.; Hunt, J. E.; and Wilson, M. S. 1997. Combining functional and structural reasoning for safety analysis of electrical designs. *Knowledge Engineering Review* 12(3):271–287.

Price, C. J.; Snooke, N. A.; and Ellis, D. J. 1999. Identifying design glitches through automated design analysis. In *Invited paper in Innovative CAE track, Procs 44th Annual Reliability and Maintainability Symposium.* RAMS.

Price, C. J.; Snooke, N.; and Landry, J. 1996. Automated sneak identification. *Engineering Applications of Artificial Intelligence* 9(4):423–427.

Snooke, N. A. 1999. Simulating electrical devices with complex behaviour. *AI Communications special issue on model based reasoning* 12(1-2):44–59.

Spangler, C. S. 1999. Equivalence relations within the failure mode and effects analysis. In *RAMS 99.* IEEE.