Validation of In-Vehicle Real-Time Applications

Nicolas Navet *, Ye-Qiong Song

LORIA - INPL

TRIO Team

ENSEM - 2, Avenue de la forêt de Haye

54516 Vandoeuvre-lès-Nancy - France

{nnavet,song}@loria.fr

Abstract

This paper proposes a pragmatic approach for validating, at the design step, real-time in-vehicle applications using CAN (Controller Area Network) as the underlying communication system. By validation, we mean the verification that dependability constraints (e.g. deadlines, jitters, transmission error tolerance) will be met by the designed solution (called the operational architecture and denoted by OA). Our approach consists, on the one hand, in building models (both analytic and simulation) and analysing them (model-based evaluation), and on the other hand, in monitoring the network(s) on a prototype of the system (prototype-based evaluation). The proposed method, as well as the set of associated software tools, is described with a special emphasis on analytical models.

Key words: In-Vehicle Application, CAN, Validation, Design, Performance Evaluation.

* Corresponding author.

1 Introduction

Distributed real-time applications such as process control and embedded systems are often time-critical and safety-critical, thus requiring a particular effort in the validation of the Operational Architecture (OA) at the design step. In this paper, we are interested in the validation of in-vehicle embedded systems using CAN (Controller Area Network) as the communication system to interconnect the distributed Electronic Control Units (ECU).

Electronic units began to equip vehicles in the seventies, and have increased the need for real-time communications within a vehicle. The use of more dedicated signal lines soon became impossible because of cost, reliability and repair problems (the total length of cables in some cars released in the 80's is over 2km with a total weight above 100kg, reported in Lawrenz (1997)). To fulfill this urgent need for multiplexed communication, 'Robert Bosch GmbH' designed the CAN network.

2 The CAN network

2.1 The CAN protocol

CAN is a broadcast bus, with priority based access to the medium and a non-destructive collision resolution. Nodes do not possess an address and no single node plays a preponderant role in the protocol. Each message has an identifier, unique to the whole system, that serves two purposes : assigning

a priority for the transmission (the lower the numerical value, the greater the priority) and allowing message filtering upon reception. Data, possibly segmented in several frames, may be transmitted periodically, sporadically or on-demand. Note that there are 2 versions of CAN differing in the size of the identifier: CAN 2.0A with an 11 bit identifier and CAN 2.0B with a 29 bit identifier. In this paper, we will focus on CAN 2.0A which provides a sufficient number of identifiers for in-vehicle communications. A minimal CAN communication profile consists of a three-layered architecture : physical layer, Data-Link Layer (DLL) and application layer. The DLL is implanted in an electronic component called a CAN controller. The ISO standards (ISO (1994b) and ISO (1994a)) only define the physical layer and DLL, but proposals have been made for the application layer (CAN Application Layer - CAL see CiA (1995)) or for complete profiles based on the two normalized layers (Smart Distributed Systems - SDS see CENELEC (1997), DeviceNet see Allen-Bradley (1994) or CANopen which uses a subset of CAL see CiA (1996)). The reader interested in more detailed information on CAN should refer to Kiencke and Kytölä (1996), Lawrenz (1997), ISO (1994b) and ISO (1994a).

CAN has very efficient error detection mechanisms. In Unruh *et al.* (1989) the authors have estimated the expected number of undetected transmission errors during the lifetime of a vehicle to be lower than 10^{-12}, that is why we will further assume that all errors are correctly detected. Each station which detects an error sends an "error flag" which is a particular frame composed of 6 consecutive dominant bits (in CAN's terminology, the dominant bit value is "0" while "1" is said the recessive bit value) that enables all the stations on the bus to be aware of the transmission error. The corrupted frame automatically re-enters into the next arbitration phase, which can lead it to miss

its deadline. The error recovery time, defined as the time from detecting an error until the possible start of a new frame, is 17 to 31 bit times (where the bit time is the time between the emission of two successive bits of the same frame). To prevent a defective node from perturbing the functioning of the whole system, for instance by repetitively sending error frames, the CAN protocol includes fault confinement mechanisms whose objectives are (1) to detect permanent hardware dysfunctioning and (2) to switch off defective nodes. For this purpose, a CAN controller has to possess 2 distinct error counters :

- the Transmit Error Counter (TEC) which counts the transmission errors detected on the frames that the station sends,
- the Receive Error Counter (REC) which counts the transmission errors detected on the frames that the station receives.

Each time a frame is correctly received or transmitted by a station, the value of the corresponding counter is decreased (except when the value is already zero). Similarly, each time a transmission error is detected, the value of the corresponding counter is increased. Depending on the value of both counters, the station will be in one of the 3 states defined by the protocol :

- *Error Active* (REC<128 and TEC<128) : the station can normally send or receive frames. This is the default state at controller initialization.
- *Error Passive* (REC>127 or TEC>127) and TEC<=255) : the station may emit but it must wait 8 supplementary bits after the end of the last transmitted frame. Furthermore, the station is not allowed to send an *active error flag* upon the detection of a transmission error, instead it will send a *passive error flag* which is made of 6 recessive bits and has thus no influence on the electric level of the bus. In this state, because of the 8 supplementary

bits before sending, the frames sent by the station are no longer certain to respect the worst-case response times computed through schedulability analysis (see paragraph 4.1).

- *Bus-off* (TEC>255) : The station is automatically switched off from the bus. In this state, the station can neither send or receive frames. A node can leave the bus-off state after a hardware or software reset (*normal mode request*) and after having successfully monitored 128 occurrences of 11 consecutive recessive bits (a sequence of 11 consecutive recessive bits corresponding to the ACK, EOF and the intermission field of a data frame that has not been corrupted).

The conditions for changing from one state to another are summarized in figure 1 :

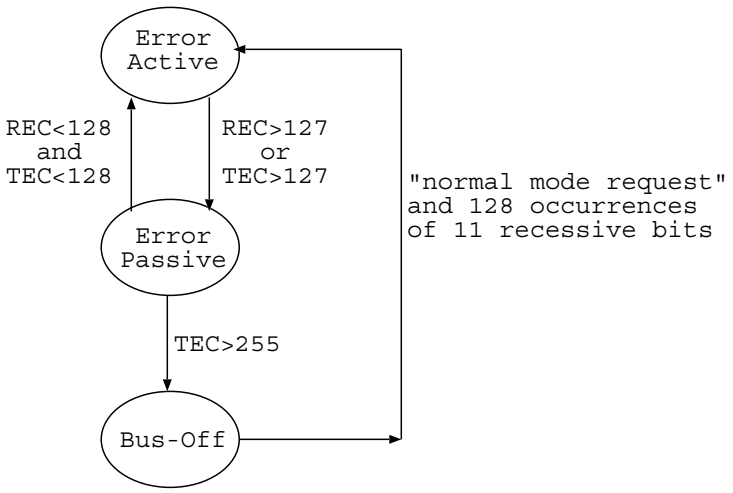


Fig. 1. The different possible states of a CAN station.

The rules for increasing and decreasing the TEC and the REC of a station are somewhat complex, see ISO (1994b) pages 48-49. In the rest of the article, we will assume that no errors occur during the signalling of an error (no bit error in an active error flag). Furthermore, we will not consider two exceptions to the general rules listed below that are useful during the initialization phase

of the system (see Lawrenz (1997) page 93) where only one node may be on-line. Given these assumptions, the rules for modifying the counter value of the stations are :

(1) frame transmission successful :
- The node is not the sending node : if the REC is between 1 and 127, then it is decreased by one. If the REC's value is nil, it stays unchanged. Finally, if its value is greater than 127, it randomly takes a value between 119 and 127.
- The node is the sending node : if the TEC is not nil, this is decreased by one, otherwise it remains unchanged.

(2) Unsuccessful transmission (transmission error detected) :
- The node is not the sending node : The REC is increased by one.
- The node is the sending node: the TEC is increased by 8.

Whatever the result of transmission, there is no more than one counter whose value is modified on a given station.

2.2 CAN as an on-board vehicle network

The current trend in the design of vehicle multiplexing systems is to interconnect 2 sub-networks :

- One for the real-time control of the vehicle with nodes such as the engine controller and the AGB (Automatic Gear Box). Typically, the data transmission rate for this network is $\geq$ 250Kbit/s and it interconnects 6 to 16 nodes.
- One for information sharing in the bodywork of the vehicle. The transmis-

sion rate for this network is usually lower (typically 125Kbit/s). Note that the network for the bodywork is not necessarily CAN - it could also be the J1850 (SAE, 1996) or VAN (Vehicle Area Network - a French standard) see ISO (1995).

In this study, we will focus our analysis on the performance of the "real-time control" sub-network because it's the one that transmits information with the more stringent constraints. Throughout this paper, to illustrate our approach, we take as example an experimental embedded CAN-based system proposed by PSA (Peugeot-Citröen) Automobiles Company. The system under study is composed of a set of 12 periodic messages listed in Table 1. There are six devices exchanging messages : engine controller, wheel angle sensor, AGB, ABS (Anti-Lock Braking System), device y (the name of this device can not be communicated because of confidentiality) and the bodywork network gateway. The transmission rate of the CAN bus is 250kbit/s and jitters are neglected (i.e. variability in queueing times). The deadline of each frame, which is the longest tolerable time between the release of a message and its delivery to all the recipients, is equal to its period (duration between two successive productions of the message). The Data Length Code (DLC) denotes the number of data bytes of each frame. Note that for safety purposes, the sources of critical information can be replicated, for example, the speed of the vehicle is produced and broadcast by both the engine controller and the ABS.

Priority	Transmitter node	DLC	Period
1	engine controller	8	10 ms
2	wheel angle sensor	3	14 ms
3	engine controller	3	20 ms
4	AGB	2	15 ms
5	ABS	5	20 ms
6	ABS	5	40 ms
7	ABS	4	15 ms
8	bodywork network gateway	5	50 ms
9	device y	4	20 ms
10	engine controller	7	100 ms
11	AGB	5	50 ms
12	ABS	1	100 ms

Table 1

Set of messages transmitted on the CAN network.

2.3 Dependability constraints

As with all process control systems, the prime function of an in-vehicle embedded system is to diagnose, via the sensors, the controlled process state and to follow its evolution in real-time. According to the diagnosis, the control devices act upon the controlled process by means of the actuators to achieve

a given objective. Two diagnostic methods can be distinguished : time triggered and event triggered, which respectively generate periodic and aperiodic traffic. Most of the transmitted information is of limited time validity, from the communication point of view this means that the response time of a message, defined as the longest time between the release of the task queueing the message and the latest time that the message arrives at the destination processor(s) must be upper bounded.

More complex time constraints exist at the application level. For instance, the response to a stimulus event must be done within a given time window. A typical example in a vehicle is the delay between the gear change (stimulus event) and the engine torque change (response to the stimulus - see Figure 6).

An in-vehicle embedded system is considered to be safety-critical because its correct functioning is vital to the safety of the passengers. Dependability constraints must thus be identified (Ziegler *et al.*, 1994) and their satisfaction verified at the design step. Among the numerous dependability constraints, we will focus in this paper on the reliability of the transmission. Due to cost pressure, the transmission support generally used in a vehicle is an unshielded twisted pair, which provides very incomplete immunity to electromagnetic disturbances (see Barrenscheen and Otte (1997) for more details on the electromagnetic sensitivity of different types of transmission support).

2.4 Factors disturbing the respect of dependability constraints

As told above, in-vehicle embedded systems suffer from strong EMI (electromagnetic interferences) which represent a serious threat to the correct behaviour of the system. The EMI (Noble, 1992; Zanoni and Pavan, 1993) can

either be radiated by some in-vehicle electrical devices (switches, relays..) or come from a source outside the vehicle (radio, radars, flashes of lightning..). We will only focus on the effect of EMI on the data transmission because the transmission support is a particularly "weak link", but EMI could also affect the correct functioning of all the electronic devices of the vehicle. On CAN, under given EMI conditions, the frequency of errors depends not only on the bus transmission rate and on its electrical characteristics, but also on the transmission support : measurements have shown the untwisted/unshielded pair to be about 6 times more sensitive to EMI than the twisted/shielded pair (Barrenscheen and Otte, 1997). The use of an all-optical network, which offers very high immunity to EMI, is not yet feasible because of the low-cost requirement imposed by the automotive industry.

On CAN, a corrupted frame automatically re-enters into the next arbitration phase, which can lead it to miss its deadline. Note that any higher priority frame that became ready since the original frame won arbitration will gain the bus after the error recovery time, thus possibly extending the response time of the original frame by far more time than the error recovery time.

CAN possesses fault confinement mechanisms, summarized in paragraph 2.1, that are aimed at preventing local disturbances (e.g. a defect transceiver, defect wire bound due to vibrations or thermo-mechanical stress, see Zanoni and Pavan (1993)) from perturbing the whole network from correctly functioning. Imagine a controller that would detect all frames on the network to be corrupted, it will repetitively send error flags that will prevent any information from being exchanged on the bus. Such fault confinement mechanisms are aimed at detecting permanent faults, however under severe EMI conditions, one or several nodes can be switched off from the bus (bus-off state) just because of successive transmission errors. For the application designer, it

is important to estimate the probability of such events to determine whether it is necessary or not to take those events into account (for instance with a mode change on the disconnection of a station).

Moreover, some nodes, for instance provided by third-party suppliers, may not respect the assumptions made on the traffic that they generate (period variation, message size variation, priority variation, transmission of unknown frames...). Such nodes that do not respect the rules have been dubbed "babbling idiots" in (Tindell *et al.*, 1995). Alterations of the traffic characteristics can lead the system not to respect the timing behaviour predicted by analytic and simulation models. Tindell and Hansson (1995) have proposed a hardware solution for this "babbling idiot" problem requiring some changes at the communication controller level but which have, to the best of our knowledge, not yet been implemented.

3 Methods and Tools for Validating CAN-Based Applications

The validation process that is proposed in this study is carried out on the Operational Architecture (OA) at the design step of the application. The OA is defined in (Simonot-Lion *et al.*, 1995) as the result of the mapping of the Functional Architecture (FA) onto the Physical Architecture (PA). The FA is the result of the specification step expressed in a formal way (Simonot-Lion *et al.*, 1995). It describes the elementary functions that the system must provide disregarding its PA. The PA is the set of devices (e.g. computers, actuators, sensors, ECU) and networks on which we intend to implement our FA (with a certain distribution strategy). Note that for in-vehicle applications, the PA is subject to very strong economic constraints. The result of the design

step should be a validated and optimal OA. The tools that we present here aim to help the designer to conceive and validate the OA.

For this purpose, we use two approaches :

- *Model-based evaluation*, both analytic and simulation models.
- *Prototype-based evaluation* by monitoring the behaviour of a prototype of the vehicle.

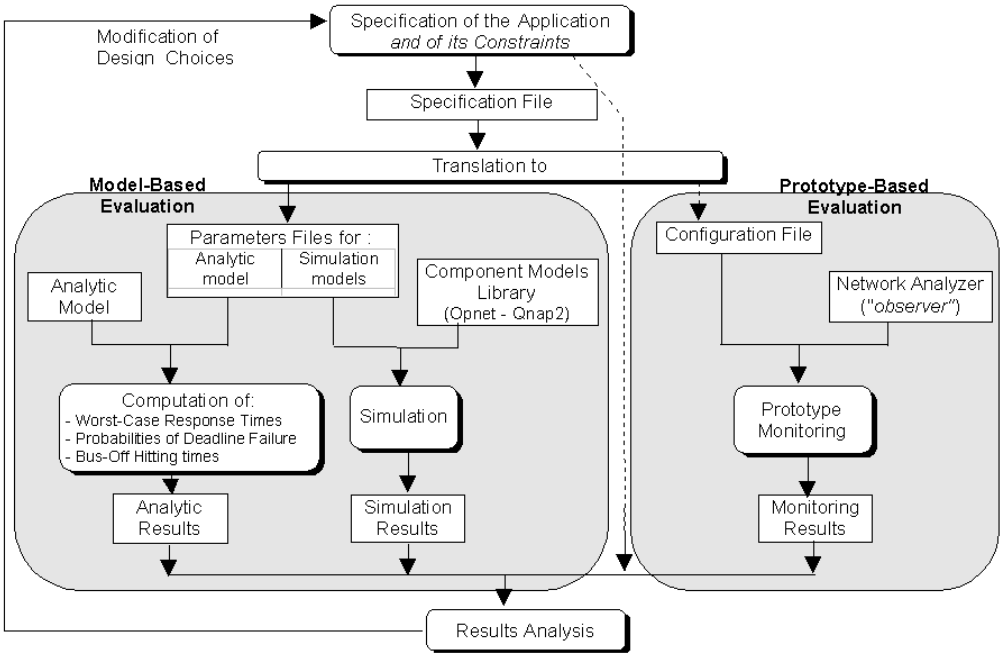


Fig. 2. Framework of the validation process

Figure 2 shows the framework of the validation process. The application is specified in terms of tasks, data exchange between tasks (messages) and distribution of tasks over nodes. The reader interested in the specification language should refer to (Navet and Song, 1997). After having checked the syntactic and lexical correctness of the specification using *lex* and *yacc*, the specification is translated to parameter files for analysis and for simulation models : *Opnet* (MIL3, 1995) and *Qnap2* (Simulog, 1995). It would be interesting to extend the specification language to application constraints to enable an au-

tomatic verification of their respect (dotted line on Figure 2). The automatic translation from specification files to network analyser configuration file is not yet feasible because our language lacks a constraints specification. After simulation, analysis and prototype monitoring, the user verifies the satisfaction of the application constraints, and, if necessary, he may modify his design choices until the system fulfills its objectives.

4 Model-based evaluation

Verification based on the OA model has several advantages compared to prototype monitoring : the existence of a prototype is not necessary (thus it could take place earlier in the development process), measurements as well as modification of system's parameters are easier. The below mentioned models were used in the context of an industrial contract with Peugeot-Citroën Automobiles (Navet and Song, 1996).

4.1 Analytical Methods

The analytic tools, written in the C language, only apply to periodic messages or to sporadic messages considered as periodic ones by taking the minimum inter-arrival time as period. They provide the worst-case response time of each frame, the worst-case deadline failure probability for each frame and the average bus-off hitting time for each station of the network. Some assumptions are made on the communication controller: (1) it must be able to send out an uninterrupted stream of frames without releasing the bus between two frames, (2) if several frames are ready to be send, it must always enter the frame of

highest priority into arbitration. It is worth noting that some earlier CAN controllers with a single transmission buffer were known not to meet these assumptions. In order to prevent the overwriting of a message, we impose that the deadline of a message m, denoted by D_m, must not be greater than its period T_m otherwise it cannot be guaranteed that a message will have been sent before the next has queued.

4.1.1 Worst-case response time calculus

The worst-case response time of message m, denoted by R_m, is defined as the longest time between the release of the task queueing message m, denoted as τ_m and the latest time that the message arrives at the destination processor(s). The message m inherits its period from τ_m but between subsequent queueing of a message there might be some variability , or jitter, because the time between the release of the sending task and the queueing time of the message depends, for instance, on the activation of higher priority tasks on the processor. We denote J_m the upper bound on the jitter (variability in queueing times) for message m. If we ignore where in the execution of task τ_m the message m is queued, then J_m is equal to the worst-case response time of τ_m. If the host processor is for instance scheduled according to the Fixed Priority Pre-emptive policy (FPP) which is possible under OSEK/VDX (real-time OS for ECUs proposed as automotive industry standard, see Kiencke and Neumann (1996); OSEK Group (1997)) then J_m can be determined using the work developed in Burns *et al.* (1995). However, it can also be evaluated when several different scheduling policies coexist on a per process basis such as in Posix1003.1b compliant OSs (Migge *et al.*, 2002).

The worst-case response time R_m must be bounded for each frame by D_m,

otherwise one can not guarantee that the timing constraints will be met, and the set of messages of the application is said to be non-schedulable. To calculate R_m, as J_m is derived from the scheduling of tasks and as the transmission time C_m can be upper bounded (see equation 3), one just has to compute the maximum time needed by the message to win the arbitration (this time is termed the "interference" time and denoted by I_m). A message m can be delayed by higher priority messages and by a lower priority message that has already obtained the bus (this time denoted as B_m is the transmission time of the longest lower priority message). Thus, one has (Tindell and Burns, 1994b) :

$$R_m = C_m + J_m + I_m \tag{1}$$

where I_m is the interference time, i.e. the longest time that all higher priority messages can occupy the bus, plus B_m (Tindell and Burns, 1994b) :

$$I_m^n = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{I_m^{n-1} + J_j + \tau_{bit}}{T_j} \right\rceil C_j \tag{2}$$

with τ_{bit} the bit time, $hp(m)$ the set of messages of higher priority than m, T_j the period of message j and C_j the bound on the transmission time of message j with d_j data bytes :

$$C_j = \left(47 + 8d_j + \left\lfloor \frac{34 + 8d_j - 1}{4} \right\rfloor \right) \tau_{bit} \tag{3}$$

I_m is computed starting with $I_m^0 = 0$ until convergence. The reader might refer to Tindell and Burns (1994b) and Tindell *et al.* (1995) for more details. The upper curve on Figure 5 shows the worst-case response times for our application.

This response time analysis has been extended in Tindell and Burns (1994a) and Tindell *et al.* (1995) by considering the possibility that errors can occur

on the bus with the following error model :

- during a time t, at most one burst of errors may occur whose size is n_{error}.
- Except for this burst of errors, the error period is at least T_{error}.

The number of transmission errors during time t can thus be bounded by $\left(n_{error} + \left\lceil \frac{t}{T_{error}} \right\rceil - 1\right)$ The response time for frame m with the above error model is obtained by changing equation 2 as follows :

$$I_m^n = E_m(I_m^{n-1} + C_m) + B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{I_m^{n-1} + J_j + \tau_{bit}}{T_j} \right\rceil C_j \qquad (4)$$

with $E_m(t)$ the error recovery overhead function which gives an upper limit to the overheads due to error recovery that could occur in an interval of duration t :

$$E_m(t) = \left(n_{error} + \left\lceil \frac{t}{T_{error}} \right\rceil - 1\right) \left(O_{err} \cdot \tau_{bit} + \max_{\forall j \in hp(m) \cup \{m\}} C_j\right) \qquad (5)$$

where O_{err} is the error recovery time (see paragraph 2.1) that is considered equal to 23 bits in the rest of the paper (23 bits being the maximum overhead with an "error active" transmitting node, see ISO (1994b)).

In the case of in-vehicle applications where the bus perturbation level may greatly vary over time, we are at present unable to give a bound to the number of errors that may occur during a given time interval. In our opinion, a random phenomenon such as transmission errors can be more accurately modeled using a stochastic process.

4.1.2 Worst-case deadline failure probability calculus

Considering that a 100% reliable medium is unrealistic in the face of the reality of in-vehicle multiplexing applications, and given that it is not always possible to bound the number of errors that may occur during a given time period, we

have developed an analytic method aimed at evaluating the probability that a message fails to meet its deadline .

For this purpose, we propose a flexible error model making assumptions on error frequency and gravity. Error occurrence obeys a Poisson Process which is denoted by $N(t)$. When an error occurs, it is either a burst of errors or a single error (respectively with probability α and $1-\alpha$).

For in-vehicle network, single and burst errors respectively correspond to a normal and to a strongly disturbed area (such as the surroundings of an airport that are scanned by radars). In fact, strong disturbance appears as the inaccessibility of the medium during a time period, leading to systematic transmission errors followed by the corresponding retransmissions.

We denote y_i the discrete random variable (r.v.) giving the error size (single error and burst errors) and u the r.v. giving the burst's size. From a mathematical point of view, this model can be seen as a Generalized Poisson Process (GPP), denoted by X(t), counting the number of errors over time. We have thus :

$$X(t) = \sum_{i=0}^{N(t)} y_i \qquad \text{with } y_0 = 0 \tag{6}$$

where:

- $N(t)_{t\geq 0}$ obeys the Poisson law of parameter λ with $P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$
- $y_i = \begin{cases} u, \text{ with probability} \quad \alpha \\ 1, \text{ with probability } 1-\alpha \end{cases}$

u is the discrete r.v. giving the burst size and y_i the r.v. giving the error size (single error and burst errors). The value of parameter λ, the probability distribution function (p.d.f) of u and α (and thus, the p.d.f. of y_i) should

be set according to the perturbations on the bus. This can be done with the experience gained with similar systems or with measurements carried out on board of prototypes. Note that some new CAN controllers possess features that facilitate this task such as readable error counters (NEC controllers, see Hausmann and Gebing (1997)) or better, have the possibility of triggering an interrupt when an error occurs (Philips controllers, see Hank (1997)). These features could even help us to conceive on-line adaptive scheduling policies (see Navet and Song (1999b,a) for a first step in that direction).

Starting from the response-time analysis with transmission errors made in Tindell and Burns (1994a); Tindell *et al.* (1995), we propose then to compute, for each frame m, the maximum number of transmission errors (denoted by K_m) under which the deadline is still respected. $E_m()$, the error recovery overhead function which was introduced by Tindell et al. as a function of time becomes here a function of the number of errors n :

$$E_m(n) = n \cdot (23\tau_{bit} + max_{\forall j \in hp(m) \cup \{m\}} C_j) \tag{7}$$

The calculus of the response time for frame m with K_m errors, denoted by $R_{m_{max}}$, is performed iteratively, starting with $K_m = 0$ and incrementing this at the end of each step. This continues until $R_{m_{max}} > D_m$ with $E_m()$ of equation 4 now defined by equation 7. In order not to take account of the last unsuccessful iteration, we have then to subtract 1 from K_m. If less than K_m errors occur during $R_{m_{max}}$, the frame m will necessarily meet its deadline, otherwise it could miss it. The Worst-Case Deadline Failure Probability (WCDFP) is the

probability that more than K_m errors occur during time $R_{m_{max}}$:

$$P[X(R_{m_{max}}) > K_m] = 1 - \sum_{k=0}^{K_m} P[X(R_{m_{max}}) = k] \quad (8)$$

It is termed worst-case because an underlying assumption is that the time needed by the frame m to gain the bus, at each retransmission, is the maximum possible (see equation 4) and that each error is detected at the last bit of the transmission and induces the longest error recovery time (see equation 7).

The classical way to compute $P[X(t) = k]$ is to differentiate k times the GPP generating function $X(z)$ (see Parzen (1962)) : $P[X(t) = k] \doteq \frac{1}{k!} \frac{d^k X(z)}{dz^k}|_{z=0}$. In practice, the generating function is not always easy to obtain and its derivative may tend rapidly to become so big that it is impossible to handle it with symbolic calculus software. Moreover, when attempting such an approach, we were faced with severe numerical accuracy problems. These issues led us to conceive a new method of calculus by using the fact that our stochastic process can be seen as a Markov Chain. This method solves the derivative size and accuracy problems and also does not need a generating function: it can cope with burst-size frequency histograms, coming directly from measures carried out on prototypes which from a practical point of view is important. This calculus method is described in Appendix A.

Figure 3 shows the WCDFP for each of the 12 frames of our application for $\lambda = 10$ and $\lambda = 30$ with $\alpha = 0.1$. The distribution of u is defined by $P(u = k) = kp^2q^{k-1}$ with $q = 1 - p$ and $p = 0.04$. The WCDFP provides useful knowledge on the system's reliability which is considered as the main dependability indicator Ziegler *et al.* (1994). The WCDFP can help the de-

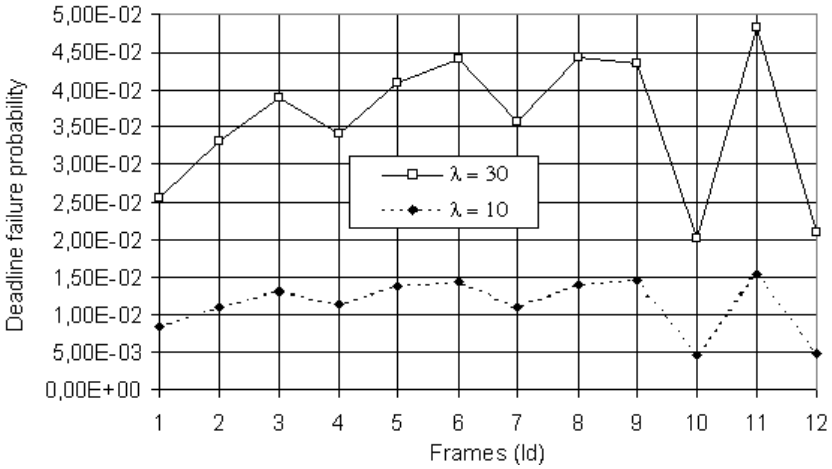


Fig. 3. Worst-case deadline failure probability for the 12 frames of the application (taken from Navet *et al.* (2000)).

signer to set the periods and priorities of messages. It can also help him choose the right transmission support regarding its dependability and cost objectives. For more details, the reader might refer to Navet and Song (1998); Navet *et al.* (2000).

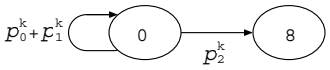
4.1.3 Bus-off hitting time

CAN fault confinement mechanisms (see paragraph 2.1) are conceived to disconnect defective nodes from the network and prevent them from perturbing the whole network. However, under severe EMI conditions, one or several nodes can reach the bus-off state just because of transmission errors. It is thus important to estimate the probability of such events which can be achieved through the knowledge of the average hitting time of the bus-off state and of the variance of the bus-off hitting times. Under the assumption that state changes are exponentially distributed, the evolution of the TEC can be modeled by a Markov chain that that for convenience we chose in discrete times (see (Parzen, 1962) for a classification of Markov chains). We thus have to

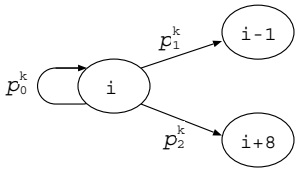
hypothesize that the time is discrete, the slot time being the average transmission time of a message sent by the considered station. This comes down to imposing that a station can only take the bus at the beginning of a time slot, which, over a long range of time, does not change the number of messages actually transmitted.

Let p_0^k be the probability that the station k has no message to transmit on a given slot time, p_1^k the probability that the station transmits a message that will not be corrupted and p_2^k the probability that the message transmitted by the station will be corrupted. The general rule is that the TEC value is increased by 8 on the transmitting node if a frame is corrupted and that the TEC is decreased by 1 if the transmission is successful. Nevertheless different cases have to be distinguished depending on the TEC value (denoted by i) :

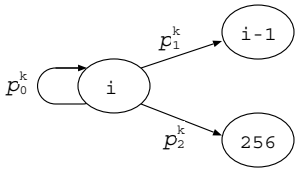
- $i = 0$:



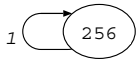
- $i \in \{1..248\}$:



- $i \in \{249..255\}$:



- $i = 256$:



The computation of p_0^k, p_1^k and p_2^k is detailed in Appendix B.

The state 256, which corresponds to the switching of the station into the bus-off state, is a so called "absorbing state" from which it is impossible to

escape and that stops the process. This is exactly the functioning scheme of the CAN protocol, when a station becomes "bus-off", it can neither send or receive frames. With the previously exposed rules, one obtains the following probability transition matrix of size $257 * 257$ (the Markov Chain having 257 states) :

$$\mathcal{P} = \begin{array}{c|cccccccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & .. & 253 & 254 & 255 & 256 \\ \hline 0 & p_0^k + p_1^k & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_2^k & 0 & .. & 0 & 0 & 0 & 0 \\ 1 & p_1^k & p_2^k & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_2^k & .. & 0 & 0 & 0 & 0 \\ 2 & 0 & p_1^k & p_2^k & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . & .. & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & .. & . & . & . & . \\ 254 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & p_1^k & p_0^k & 0 & p_2^k \\ 255 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 0 & p_1^k & p_0^k & p_2^k \\ 256 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 0 & 0 & 0 & 1 \end{array}$$

The matrix $\mathcal{P}$ can be transformed in its "canonical form" :

$$\mathcal{P} = \begin{bmatrix} 1 & 0 \\ \mathcal{R} & \mathcal{Q} \end{bmatrix} \tag{9}$$

where $\mathcal{Q}$ is the original matrix without the 257^{th} line and the 257^{th} row, $\mathcal{R}$ is the 257^{th} column vector of $\mathcal{P}$ without the 257^{th} element. For all stochastic matrices which can be partitioned in the same form as $\mathcal{P}$ (see equation (9)),

the matrix

$$\mathcal{S} = \sum_{r=0}^{\infty} \mathcal{Q}^r = (I - \mathcal{Q})^{-1} \tag{10}$$

exits (the determinant of $\mathcal{Q}$ is not nil). This matrix plays an important role in determining the average hitting time and the variance of the hitting times. We note $\mathcal{S}_\rho$ the row vector whose k^{th} element is the sum of the elements of the k^{th} line of $\mathcal{S}$. In a similar way, $\mathcal{S}_{\rho 2}$ is the row vector whose k^{th} element is the square of the k^{th} element of $\mathcal{S}_\rho$. It has been proven in Bhat (1984) (pages 77-79) that $||E(N_i)|| = \mathcal{S}_\rho$ where N_i is the random variable which gives the time needed to reach for the first time the absorbing state starting from a given state i. Thus $||E(N_i)||$ is the row vector which gives the expected value of the first hitting time starting from state i. Similarly, one has $||V(N_i)|| = (2\mathcal{S} - I)\mathcal{S}_\rho - \mathcal{S}_{\rho 2}$ with $||V(N_i)||$ the row vector which gives the variance of the hitting times.

To illustrate this analysis, one considers two stations : the "engine controller" and the "bodywork network gateway" which respectively send the frames of priority $\{1, 3, 10\}$ and $\{8\}$ (cf. table 1). The average size of the frames for the engine controller is 118.75 bits while being 105 bits for the bodywork network gateway. On figure 4, one can observe that the average hitting time greatly varies depending on the BER. For instance, it takes in average about 40 seconds for the engine controller to reach the bus-off state with a Bit Error Rate (BER) of 0.001 (corresponding to a frame error rate of about 11.17%) and more than 43360 hours with a BER of 0.0007 (to be compared to the expected cumulated utilization time of a vehicle which is about 5000 hours). In addition, the curves on figure 4 suggest to us that the more important the load induced by a station, the more frequently the station will reach the bus-off state (the load generated by the engine controller is 7.6% versus 0.84% for the bodywork network gateway). It is also noteworthy that the standard

deviation of the hitting times is very important, it is of the same order of magnitude than the average hitting times which in practice means that there will be a high variability among the observed hitting times.

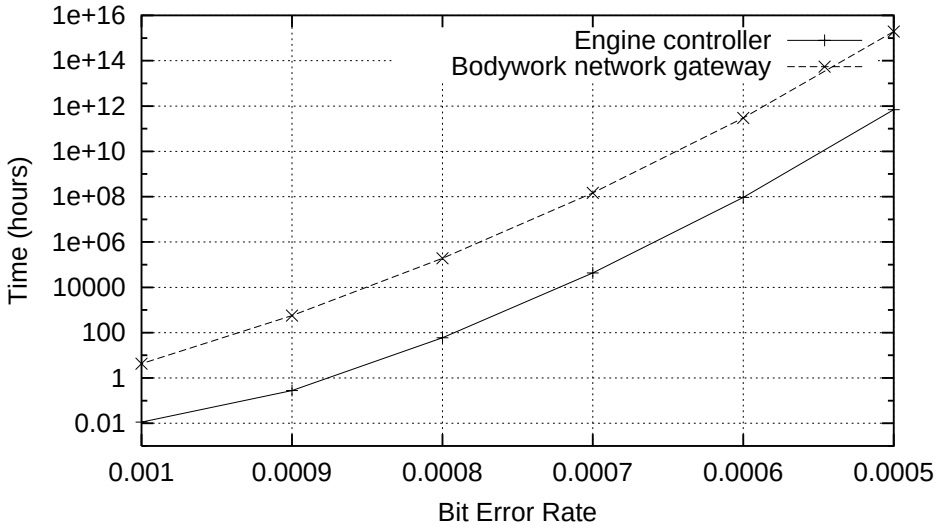


Fig. 4. Average hitting time of the bus-off state for the engine controller and the bodywork network gateway with the Bit Error Rate (BER) varying from 0.001 to 0.0005 .

4.2 Simulation

We have built two simulation models using two different modeling formalisms : queueing networks for *Qnap2* (Simulog, 1995) and finite state machines with embedded C statements for *Opnet* (MIL3, 1995). These models were, for the main part, implemented by two different persons. The verification (i.e. correctness of the implementation) and validation (i.e. closeness of the model output to the real system) of simulation models is always a problematic step, to diminish the unavoidable uncertainties regarding simulation model quality, the outputs of both models were thoroughly compared. The reader may refer to Jagdev *et al.* (1995) for a very complete discussion on verification and

validation of models.

4.2.0.1 The *Opnet* Model is composed of an application layer model, a task model and a communication controller model (Intel 82527). These simulation models are actually created using *Opnet External Model Access* library. By default, stations are placed every meter on a linear medium, the topology of the system can be modified using the *Opnet* graphical editor before the simulation. The model is detailed in (Navet and Song, 1997) and it was available through *Delta-Partners* (former French dealer of *Opnet*).

4.2.0.2 The *Qnap2* model is composed of 3 objects : application layer, communication controller (Intel 82527) and a physical layer object. The *Qnap2* model is much simpler than the *Opnet* one and for instance, it does not take into account the topology of the bus. A set of macro-instructions for creating objects and building communication profiles (interconnection of objects through service access points) has been written. This model is described in (Navet, 1996).

The results obtained from the simulation models include the mean and maximum response time as well as the average deadline failure rate under unreliable transmission with the error model defined by equation 6. As an illustration of the results obtainable through simulation, Figure 5 represents the maximum and mean response times which we have recorded. We choose to illustrate uniquely the results of *Opnet* as those obtained through *Qnap2* were extremely close.

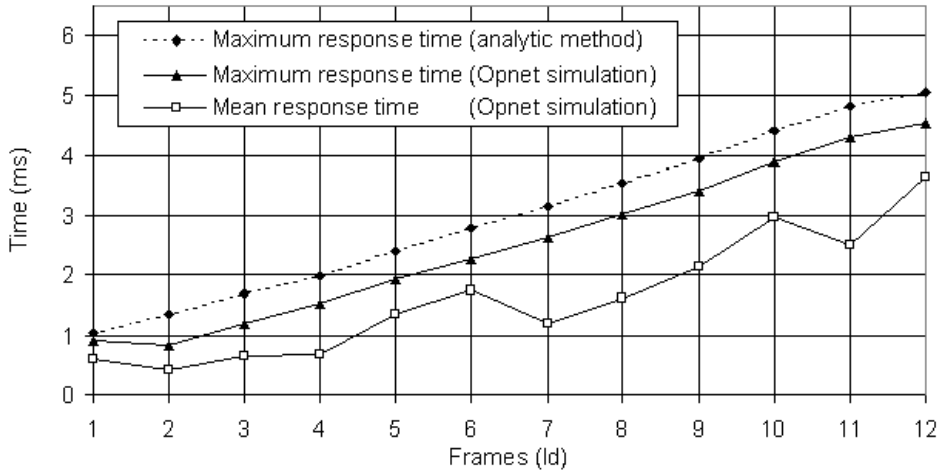


Fig. 5. Response time estimation using analysis and simulation.

Figure 5 also represents the analytical worst-case response time which is, for all frames, lower than the deadline so that the set of messages is guaranteed to meet its constraints with reliable transmission. Logically, the analytical results always upper bound the maximum and the mean values collected during simulation. The simulations were performed under the assumption that all nodes are synchronized (they are all assumed to begin transmitting from the start of the simulation) which explains that the curve of the mean response times is not smooth.

5 Prototype-based evaluation

In the automotive industry, the construction of prototypes is an integral part of the design activity. By monitoring a prototype, one could get useful information which help the validation of models as well as the tuning of their parameters.

In the context of an industrial contract with Peugeot-Citroën Automobiles (Song and Navet, 1997), we developed a network analyzer, named *"Observer"*, that

captures all frames on the network and verifies on-line the respect of the traffic characteristics as well as some specified temporal properties. We have conceived a small configuration language (Song and Navet, 1997) to specify:

- Messages of the application (identifier, data length)
- Signals to be displayed on the screen (speed, engine's rpm, torque ...)
- Application level constraints to be verified (stimulus-response delay)

The results are displayed and also logged into a file. To our surprise, we may occasionally observe the appearance of some unspecified frames on the network in particular circumstances (e.g. initialisation). These frames may disturb the correct operation of the system because of a network load increase. In fact, most of the ECUs of a vehicle are provided by third-party suppliers and these devices may not fulfill their specification. By default, "unknown" frames are recorded and period variations (observed upon reception) analyzed (average value, maximum value, standard deviation) for all messages of the application.

"Observer" is written in the C language and runs on a portable PC equipped with a PCMCIA CANcard controller (Softing GmbH, 1995). Much care was taken to optimize the execution speed of some part of the code because the number of frames to capture and analyze each second may be large : on a 50% loaded 250kbit/s CAN network, about 1250 frames transit per second (considering an average size of 100 bits per frame). *"Observer"* was successfully used on a prototype car implementing the set of messages of Figure 1 (about 700 frames per second) being configured to verify the satisfaction of one stimulus-response delay (described in Figure 6). For more constraints and/or more frames, we should consider implementing an off-line verification by ana-

lyzing a log-file.

A typical constraint that the designer ought to verify is whether the delay between a gear change and the corresponding engine torque reduction is within a given range. In our application, the AGB periodically (15ms) sends the frame containing the gear speed. For his part, the engine controller transmits the engine torque every 10ms (see Figure 6). Once a gear change is detected, *Observer* has to analyze each incoming engine controller's frame when it notices that the engine torque drops below a specified value.

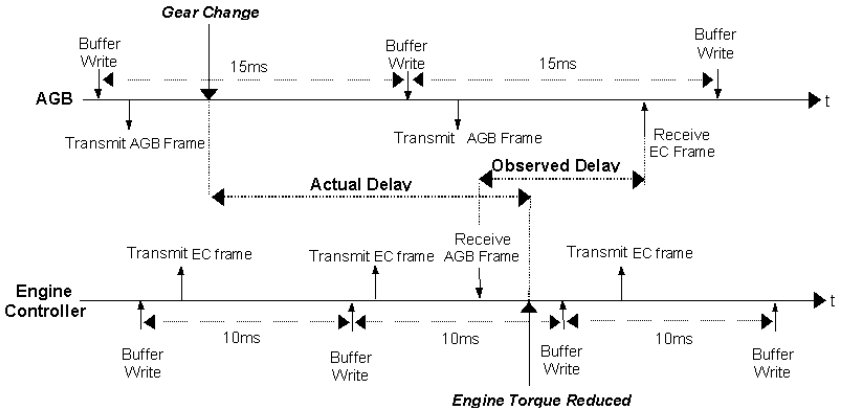


Fig. 6. Delay between a stimulus and the response (gear change - torque reduction)

It is important to realize that the Observed Delay (OD) for a stimulus-response constraint is only an approximation of the actual delay (see Figure 6). Nevertheless, this information is helpful for the application designer because it helps to upper bound the actual delay (denoted by AD), in effect, we know that :

$$AD \leq T_{agb} + J_{agb} + OD + C_{agb} - C_{ec} \tag{11}$$

where T_{agb} and J_{agb} are respectively the period and the jitter of the frame containing the gear, C_{agb} being its transmission time and C_{ec} being the transmission time of the frame sent by the engine controller.

6 Concluding remarks and future work

In this study, we presented a pragmatic approach as well as the set of associated software tools aimed at validating, during the design step, in-vehicle distributed applications using CAN. This validation is carried out by solving models (both analytic and simulation) of the operational architecture and also by prototype monitoring.

The table 2 summarizes the results one can obtain with each method and shows their complementary nature. Analytical techniques provide the bounds of considered performance metrics (worst-case response time, worst-case response time with transmission errors, worst-case deadline failure probability) or enable us to estimate the frequency of events too rare to be evaluated by simulation (average bus-off hitting time of a CAN station). On the other hand,

Constraints		**Analysis**	**Simulation**	**Monitoring**
timing constraints	worst-case response time	√		
	worst-case resp. time with trans. errors	√		
	average response time		√	
	jitter in reception			√
other dependability constraints	worst-case deadline failure proba.	√		
	average hitting time of the bus-off state	√		
application level constraint	stimulus - response delay			√
resource utilization constraint	network load	√	√	√

Table 2

Constraints and their verification methods.

simulation does not provide bounds but its results are closer to the the actual case which allows the designer to fine tune his application. Another advantage of simulation is that it is not restricted to periodic and sporadic messages, it can also deal with aperiodic traffic.

Both analytic and simulation models put the emphasis on the communication aspect but use very simplified ECU models. Although it is feasible to develop more detailed ECU models as in Courrier *et al.* (1998), this will lead to a

more complicated global OA architecture with more difficult result analysis. On the other hand, the in-vehicle application designer's main task is to integrate ECUs provided by third-party suppliers. These ECUs usually exist already at the design step of the application so why not use them instead of models ? This led us to the development of *"Observer"* which is well suited for the verification of application level constraints such as stimulus-response delay.

We envisage extending the OA specification language to application constraints, thus enabling an automatic verification of their respect. Another extension would be a better taking into account of task scheduling by modeling OSEK/VDX services such as "activation of a task upon receipt of a message" and this, by applying the holistic schedulability analysis developed in Tindell and Clark (1994) and/or by simulation. This will require the car manufacturer to have the complete knowledge of the characteristics of the tasks embedded in ECU.

One of the current challenges of the automotive industry is to define development methods and tools that will enable the reusability and the portability of application level software. For instance, up to now the same diagnostic service can be developed by several ECU manufacturers and the interoperability among the different pieces of software is a serious problem. This work of standardizing the development process of in-vehicle applications that will enable reusability and portability, and thus save costs and improve dependability, is an on-going work (in Germany, the *Softmobil* project, in France a similar project named *Architecture Electronique Embarquée* began in Sept. 1998, see http://aee.inria.fr/) involving car manufacturers, ECU suppliers and research institutes and validation is a key point for the success of those projects.

References

Allen-Bradley (1994). Devicenet specification. vol. 1 & 2.

Barrenscheen, J. and Otte, G. (1997). Analysis of the physical CAN bus layer. In *4th international CAN Conference, ICC'97*, pages 06.02–06.08.

Bhat, U. (1984). *Elements of Applied Stochastic Processes*. John Wiley & Sons. ISBN 0-471-87826-X.

Burns, A., Tindell, K., and Wellings, A.-J. (1995). Effective analysis for engineering real-time fixed priority schedulers. *IEEE Transactions on Software Engineering*, **21**(5), 475–480.

CENELEC (1997). Low voltage switchgear and controlgear - part 5: Control circuit devices and switching elements - smart distributed systems (SDS). European Committee for Electrotechnical Standardization, document CLC/TC(SEC)146 - Smart Distributed Systems.

CiA (1995). Can application layer (CAL). CAN in Automation International Users and Manufacturers Group, CiA/DS201-207.

CiA (1996). CANopen communication profile for industrial systems. CAN in Automation International Users and Manufacturers Group, CiA/DS301 (Version 3.0).

Courrier, M., Simonot-Lion, F., and Song, Y. (1998). Microscopic modeling of support system for in-vehicle embedded systems. In *International IFIP Workshop on Distributed and Parallel Embedded Systems, DIPES'98*.

Hank, P. (1997). Pelican : A new can controller supporting diagnosis and system optimization. In *4th international CAN Conference, ICC'97*, pages 04.12–04.18.

Hausmann, G. and Gebing, E. (1997). The realisation of specific automotive applications with "full" CAN functionality at "basic" CAN cost on highly

integrated 8-bit microcontroller of NEC's 78k/0 family. In *4th international CAN Conference, ICC'97*, pages 4.02–4.11.

ISO (1994a). *Road Vehicles - Interchange of Digital Information - Controller Area Network for high-speed Communication.* International Standard Organization, ISO 11898.

ISO (1994b). *Road Vehicles - Low Speed serial data communication - Part 2: Low Speed Controller Area Network.* International Standard Organization, ISO 11519-2.

ISO (1995). *Road Vehicles - Low Speed serial data communication - Part 4.* International Standard Organization, ISO 11519-4.

Jagdev, H. S., Browne, J., and P., J. (1995). Verification and validation issues in manufacturing models. *Computers in Industry*, **25**(3), 331–353.

Kiencke, U. and Kytölä, T. (1996). CAN, a ten years' anniversarial review. In *3th international CAN Conference, ICC'96*, pages 2.2–2.7.

Kiencke, U. and Neumann, K. (1996). OSEK/VDX - an open software architecture for communicating vehicle systems. In *3th international CAN Conference, ICC'96.*

Lawrenz, W. (1997). *CAN System Engineering - From Theory to Applications.* Springer-Verlag. ISBN 0-387-94939-9.

Migge, J., Jean-Marie, A., and Navet, N. (to appear in 2002). Timing analysis of compound scheduling policies : Application to Posix1003.1b. *Accepted for publication in Journal of Scheduling.*

MIL3 (1995). Opnet modeler 2.5.

Navet, N. (1996). Le Réseau CAN et sa modélisation sous forme de réseaux de files d'attente. In *Automatique, Génie Informatique, Image - AGI'96, Tours, France*, pages 293–297.

Navet, N. and Song, Y.-Q. (1996). Evaluation de performances de la messagerie

CAN du véhicule prototype PSA - action 1 du contrat PSA-CRIN. Technical report, Centre de Recherche en Informatique de Nancy (CRIN). Rapport de fin de contrat 96-R-182.

Navet, N. and Song, Y.-Q. (1997). CAN modeling : towards integrating analytic methods and simulation. In *OPnet European Users Group (OPEUG), Paris, France.*

Navet, N. and Song, Y.-Q. (1998). On fault tolerance and worst-case response time analysis in CAN. In *23nd IFAC/IFIP Workshop on Real-Time Programming, WRTP'98.*

Navet, N. and Song, Y.-Q. (1999a). Reliability improvement of the dual-priority protocol under unreliable transmission. *Control Engineering Practice*, **7**(8), 975–981.

Navet, N. and Song, Y.-Q. (1999b). Une politique à changement de priorité pour l'ordonnancement de messages dans des environnements bruités. In *Colloque Francophone sur L'ingénierie des Protocoles, CFIP'99*, pages 249–264.

Navet, N., Song, Y.-Q., and Simonot, F. (2000). Worst-case deadline failure probability in real-time applications distributed over CAN (controller area network). *Journal of Systems Architecture*, **46**(7), 607–618.

Noble, I. (1992). EMC and the automotive industry. *Electronics & Communication Engineering Journal*, pages 263–271.

OSEK Group (1997). OSEK/VDX operating system. Version 2.0 rev. 1, available at http://www-iiit.etec.uni-karlsruhe.de/~ osek/.

Parzen, E. (1962). *Stochastic Processes.* Holden-Day (ISBN 0-8162-6664-6).

SAE (1996). Class B data communications network interface - sae j1850 standard. Society of Automotive Engineers, Rev. NOV96.

Simonot-Lion, F., Thomesse, J.-P., Bayart, M., and Staroswiecki, M. (1995).

Dependable distributed computer control systems : Analysis of the design step activities. In *Proc. IFAC-DCCS'95*.

Simulog (1995). Qnap2 version 9.2 - guide de l'utilisateur.

Softing GmbH (1995). Cancard user manual (v2.03, rev.01).

Song, Y.-Q. and Navet, N. (1997). Validation d'applications distribuées autour du réseau can par vérification en-ligne - développement d'un observateur réseau. Technical report, Centre de Recherche en Informatique de Nancy (CRIN). Rapport de fin de contrat 97-R-110.

Tindell, K. and Burns, A. (1994a). Guaranteed message latencies for distributed safety-critical hard real-time control networks. Technical report, Department of Computer Science, University of York (UK). Technical Report YCS229.

Tindell, K. and Burns, A. (1994b). Guaranteeing message latencies on controller area network (CAN). In *1st International CAN Conference, ICC'94*.

Tindell, K. and Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessors and Microprogramming*, **40**, 117–134.

Tindell, K. and Hansson, H. (1995). Babbling idiots, the dual priority protocol, and smart CAN controllers. In *2nd international CAN Conference, ICC'95*, pages 7.22–7–28.

Tindell, K., Burns, and Wellings, A. (1995). Calculating controller area network (CAN) message response times. *Control Engineering Practice*, **3**(8), 1163–1169.

Unruh, J., Mathony, H.-J., and Kaiser, K.-H. (1989). Error detection analysis of automotive communication protocols. Technical report, Robert Bosch GmbH.

Zanoni, E. and Pavan, P. (1993). Improving the reliability and safety of au-

tomotive electronics. *IEEE Micro*, **13**(1), 30–48.

Ziegler, C., Powell, D., and Desroches, P. (1994). Dependability of on-board automotive computer systems. In *IEEE Intelligent Vehicles 1994 Symposium*, pages 568–575.

Appendix A: Calculus Method of $P[X(t) = k]$

According to the definition of $X(t)$ (see equation 6), we have :

$$P[X(t) = k] = \sum_{m=0}^{k} P(X(t) = k \,|\, N(t) = m) \, \frac{e^{-\lambda t}(\lambda t)^m}{m!} \tag{12}$$

- if $k = 0$, we have $N(t) = 0$ and as $y_i \geq 1$, thus $P[X(t) = 0] = e^{-\lambda t}$
- if $k \geq 1$, we have $P[X(t) = k] = \sum_{m \geq 1}^{k} P(X(t) = k \,|\, N(t) = m) \, \frac{e^{-\lambda t}(\lambda t)^m}{m!}$

Since $P[X(t) = k \,|\, N(t) = m]$ is the probability of having k errors within m package of size y_i and since $N(t)$ and the r.v. y_i are mutually independent, if we note $S_m = y_1 + y_2 + \ldots + y_m$, then :

$$P[X(t) = k] = \sum_{m=1}^{k} P[S_m = k] \, \frac{e^{-\lambda t}(\lambda t)^m}{m!} \tag{13}$$

S_m being a Markov chain, we can write :

$$P[S_{m+1} = j] = \sum_{1 \leq i \leq j} P[S_{m+1} \,|\, S_m = i] \, P[S_m = i] \tag{14}$$

We note $a_k = P(S_1 = k) = P(y_1 = k) \doteq P(y = k)$, according to the *Chapman-Kolmogorov* equation (Parzen, 1962) :

$$P[S_{m+1} = j] = \sum_{i=1}^{j} a_{j-i} \, P[S_m = i] \tag{15}$$

Noting that for $m > k$, $P[S_m = k] = 0$ since $y_i \geq 1$. Equation 15 gives us the algorithm for calculating $P[S_m = k]$. The algorithm is shown on Figure 7 where computed[,] is a two-dimensional array, used to store already computed values, whose elements must be initialised to -1.

```
1  funct real P_S(integer m, integer k)
2     real r := 0
3     if (m > k)  r := 0;
4  else if (m = 1)  r := a_k;
5     else if (computed[m, k] = −1)
6             for i := 1 to k do
7                 r := r + (a_{k−i} · P_S(m − 1, i))
8             od
9             computed[m, k] := r;
10       else r := computed[m, k];
11          fi
12       fi
13    fi
14    return r;
15 end
```

Fig. 7. Algorithm for computing $P[S_m = k]$.

Finally, according to equation 13 the algorithm for computing $P[X(t) = k]$ (with $Poisson(i, t, \lambda)$ defined as $\frac{e^{-\lambda t}(\lambda t)^i}{i!}$) is given on Figure 8

```
1  funct real compute_Xt(integer k, real t, real λ)
2     real r := 0;
3     for i := 1 to k do
4         r := r + P_S(i, k) · Poisson(i, t, λ);
5     od
6     return r;
7  end
```

Fig. 8. Algorithm for computing $P[X(t) = k]$.

Appendix B: Computation of p_0^k, p_1^k, p_2^k

p_0^k is the probability that station k does not have any message to send in the next time slot, it is equal to 1 minus the load induced by station k. One has to take account of the surcharge generated by transmission errors. To each transmission error corresponds a retransmission which can be, in its turn, corrupted (and so on). Thus we have :

$$\begin{aligned} p_0^k &= 1 - \left[\left(\sum_{m_i \in \mathcal{M}_k} \frac{C_i}{T_i} \right) + \left(\sum_{m_i \in \mathcal{M}_k} \frac{C_i}{T_i} \right) FeR_k + \left(\sum_{m_i \in \mathcal{M}_k} \frac{C_i}{T_i} \right) {FeR_k}^2 + \ \ldots \right] \\ &= 1 - \left(\sum_{m_i \in \mathcal{M}_k} \frac{C_i}{T_i} \right) / \left(1 - FeR_k \right) \end{aligned} \tag{16}$$

where $\mathcal{M}_k$ is the subset of messages sent by station k , m_i is the message of identifier i and FeR_k is the Frame Error Rate for station k which can be estimated with the Bit Error Rate, common to all the stations of the network :

$$FER_k = 1 - \sum_{m_i \in \mathcal{M}_k} \left(\frac{(1 - BER)^{S_i}}{T_i} / \left(\sum_{m_j \in \mathcal{M}_k} \frac{1}{T_j} \right) \right), \tag{17}$$

with $C_i = S_i \cdot \tau_{bit}$ and S_i is the maximal size of the message i (see equation (3)). p_1^k is the probability that station k successfully sends a frame :

$$p_1^k = (1 - p_0^k) \cdot (1 - FeR_k) = \sum_{m_i \in \mathcal{M}_k} \frac{S_i}{T_i} \tag{18}$$

p_2^k is the probability that station k sends a frame that will be corrupted during transmission :

$$p_2^k = (1 - p_0^k) \cdot FeR_k = 1 - p_0^k - p_1^k \tag{19}$$