TIMED AUTOMATA APPROACH TO CAN VERIFICATION

Jan Krakora and Zdenek Hanzalek

Czech Technical University in Prague,
Faculty of Electrical Engineering,
Department of Control Engineering
Karlovo namesti 13, Prague 2, 121 35, Czech Republic
{*krakorj,hanzalek*}*@fel.cvut.cz*

Abstract:
This article deals with verification of real time distributed system focusing on CAN model by timed automata and specification of verified properties by temporal logic. Such system, based on several CPUs, consists of an application SW running under real-time operating system (e.g. OSEK) and using standard broadcast communications based on the Controller Area Network (CAN). The crucial problem is to verify both, the time properties (e.g. message response time) and logic properties (e.g. deadlock) of such complex applications. The approach presented in this article is based on the modeling the discrete event system by Timed Automata and on verification by model checking tool (e.g. UPPAAL). In contrast to classical approaches dealing either with shared bus (guarantee message latencies approached by Tindell and Burns) or shared processor (rate monotonic analysis), our approach deals with both kinds of resources.

Keywords: Controller Area Network, Real-time, Timed automata, Medium Access Control

1. INTRODUCTION

Let us assume the distributed real time control system consisting of application processes (designed by application developer) running under Real-Time Operating System (RTOS e.g. OSEK (Geischeder *et al.*, 2000)) while using several processors interconnected via standard broadcast communication based on the Controller Area Network (CAN) (Etschberger *et al.*, 2001). Structure of the application under consideration is depicted in Figure 1. The crucial problem is to verify both, the time properties (e.g. message response time, schedulability of periodic processes, response time) and logic properties (e.g. deadlock, mutual exclusion, priority based access) of the applications incorporating two kinds of shared resources - the processor and the bus. Classical approaches deal separately either with the processor sharing (studied for example by RMS (Klein *et al.*, 1993)) or with the bus sharing (e.g. CAN message latency studied by Tindell (Tindell and Burns, 1994)).

The task schedulability on monoprocessor and multiprocessor systems is widely studied subject ((Buttazzo, 1997),(Liu, 2000)). For example Rate Monotonic Scheduling (RMS) can be used to guarantee schedulability, when the application consisting of periodic processes is running on monoprocessor with priority based preemptive kernel and the processes have their deadlines at the end of period. RMS assigns fixed priorities to the processes according to their request rate (inverse

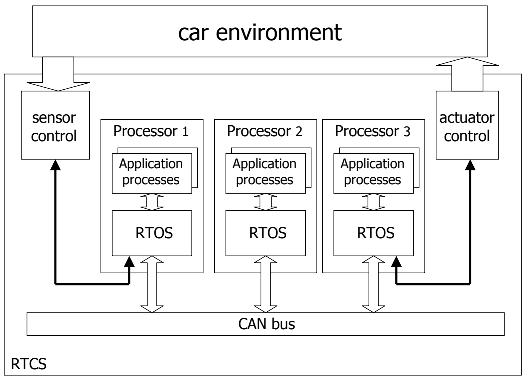


Fig. 1. Real time control system structure

to their period deadline), therefore the highest priority is assigned to the processes with highest frequency. Schedulability of such processes can be verified using Utilization bound theorem or Completion time theorem.

Prediction of the worst-case message latencies for CAN was presented by Tindell and Burns in (Tindell and Burns, 1994). This method is a direct application of the scheduling theory where the common bus is considered as shared resource. In similar way, CAN operates the fixed priority scheduling algorithm and authors assume the rate monotonic priority assignment. The message worst case response time is influenced not only by its length but also by the maximal length of one lower priority message since a high priority message cannot interrupt the message that is already transmitted. Moreover due to the priority based bus arbitration method the message worst-case response time is influenced by all higher priority messages, each of them considering their occurrence ratio.

This article presents an alternative approach based on model checking while using timed automata (Alur and Dill, 1994) and temporal logics (Katoen, 1999). Using this approach we model parts of the distributed system (application SW, operating system and communication bus) by automata. The automata use synchronization primitives enabling their interconnection. Please refer to (Berard *et al.*, 2001) on issues related to implementation and complexity of verification algorithms.

Modeling and verification of concurrent processes sharing one processor have been shown in (Waszniowski and Hanzalek, 2003), (Corbett, 1996). These works incorporate priority based preemptive and non-preemptive scheduling, intertask communication primitives and interrupt handling. These models can be directly combined with CAN model shown in this article while using synchronization primitives. That is why the operating system part is neglected and one application processes per one processor is assumed in this paper. Verification of the CAN model developed here can be directly compared to the results in (Tindell and Burns, 1994) and it can be simply enlarged to deal with internal structure of the application processes, timing parameters of different operating systems, sporadic processes etc. Moreover, while using the model checking approach, one can verify not only the schedulability, but also rather complex properties linked to logic and timing behavior of the distributed system. On the other hand the complexity is a drawback of the model checking approach in contrast to straightforward equations of the scheduling theory.

The paper is organized as follows: in Section 2 basic behavior of CAN is modeled incorporating models of transceiver, bus and application process. Sections 3 provides verification of timing and logic properties and Section 4 shows a case study with periodic and sporadic processes including comparison with Tindell's approach.

2. BASIC CONCEPT OF CAN MODEL

The aim of this section is to model CAN by timed automata. Up to the standard the CAN is a message oriented transmission protocol. Due to the bus topology only one processor can transmit at a given time. Therefore the message response time (i.e. the length of time from the release time of the message to the instant when it is completely received) is given not only by the message length, but also by the access to the shared communication media (so called Medium Access Control - MAC).

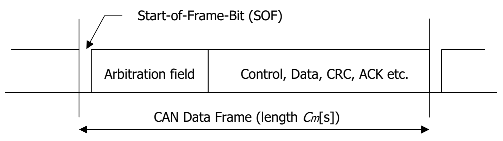
The message priority is given by the message ID. The priorities are laid down during the system design in the form of corresponding binary values and cannot be changed dynamically. The identifier with the lowest binary number has the highest priority.

MAC problem is resolved by bit-wise arbitration on the identifiers performed by each station observing the bus level bit by bit. The resolution is in accordance with the "wired and" mechanism, by which the dominant level overwrites the recessive level. The transmission is denied for all processors with recessive transmission and dominant observation. All those processors automatically become receivers of the message with the highest priority and do not re-attempt transmission until the bus is idle again.

In contrast to the results achieved in (Tindell and Burns, 1994) our approach can be simply enlarged to deal with internal structure of the application processes, timing parameters of different operating systems, and other timing and logic properties of the real time control system. Therefore it allows to check the response time, i.e. the actuator to sensor reaction time including not only the message latency but also the latencies introduced by RTOS and application processes. Moreover, while using the verification approach, one can verify not only the schedulability, but also rather complex properties linked to logic and timing behavior of the distributed system. The model is composed of timed automata described in the following subsections.

2.1 Bit-wise arbitration model

The model of CAN arbitration designed in timed automata (Pettersson and Larsen, 2000) is shown in Figure 3. The model describes MAC mechanism for one message accessing the bus. The location *no_trans_needed* represents a situation when the arbitration model is waiting for *trans_request* from the application process. The locations *send_bit_to_bus*, *listen_bus*, *check_next_bit* represent the arbitration process. The locations *request_denied* and *request_success* give result of the arbitration process.



Start-of-Frame-Bit (SOF)
Arbitration field
Control, Data, CRC, ACK etc.
CAN Data Frame (length C_m[s])

Fig. 2. CAN message frame format

After processing of the Start of Frame Bit (SOF) (see the CAN message frame format in Figure 2) the first bit from the arbitration filed is sent to the bus (transition *send_bit_to_bus* → *listen_bus*). At the same time the transmitting processor senses the bus and both transmitted bit (local variable *id*) and sensed bit (global variable *signal*) are compared. If they are identical and the end of the Arbitration field (*nsigi* states for the length of the Arbitration field) was not reached the next bit is proceeded (*check_next_bit* location) when nominal bit-time elapses (deterministically given as *tbit* constant). If the sensed bit is not identical to the transmitted one, the transmission is denied (*request_denied* location). If they are identical and the end of the Arbitration field was reached the processor wins the arbitration (*request_success* location). The CAN Arbitration model includes the information about the duration of each bit-time given by invariant $t \leq tbit$ in *listen_bus* location and guards $t \geq tbit$, $t \geq 0$ on outgoing transitions. When *tbit* is not deterministic, i.e. *tbit* is bounded on interval $\langle tbit_l, tbit_u \rangle$ then the duration of each bit-time given by invariant $t \leq tbit_u$ and guard $t \geq tbit_l$.

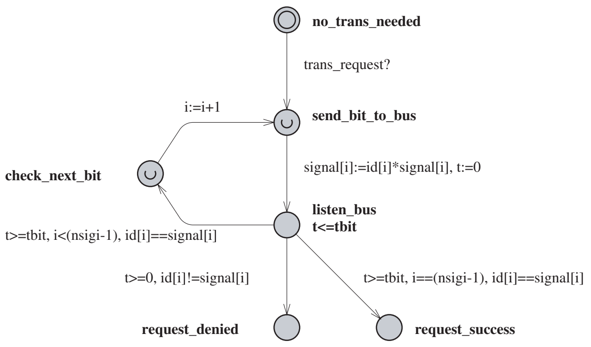


Fig. 3. Arbitration model (in UPPAAL like notation)

2.2 Transceiver model

Above explained bit-wise arbitration is a part of the transceiver model. The implementation of the complete transceiver model is depicted in Figure 4, and its interconnection with other automata is shown in Figure 6. It is composed of the three sections:

(1) the arbitration section described already in Figure 3
(2) synchronisation section ($waiting\_for\_free\_bus$ → $send\_bit\_to\_bus$ transition) that is used to synchronize all transmitting processors prior to arbitration (this part realises broadcast communication) and
(3) data transmission section given by $trans\_section$, $trans\_section\_finished$ and $trans\_finished$ locations.

The function of transceiver is the following: after receiving the transmission request, the processor is in the waiting state ($waiting\_for\_free\_bus$) until the bus is free. Bus becomes idle, the arbitration processes start (synchronization by urgent *broadcast_synch* channel). If the transmission was denied (*trans_denied* location), the transmission request is immediately repeated and the processor is waiting for free bus again ($waiting\_for\_free\_bus$ location). Otherwise the processors message is sent. The duration of message is given by deterministic time C_m. When the transmission is finished ($trans\_section\_finished$) the bus becomes idle ($bus\_trans\_finished$ channel) and the application process is informed about the end of transmission ($trans\_compl\_status$ channel).

2.3 Bus model

Figure 5 depicts the physical bus model. The model is in idle location when there is no activity on the bus and it is in busy location when any processor transmits. The *trans_vote* global variable is used to detect that at least one processor

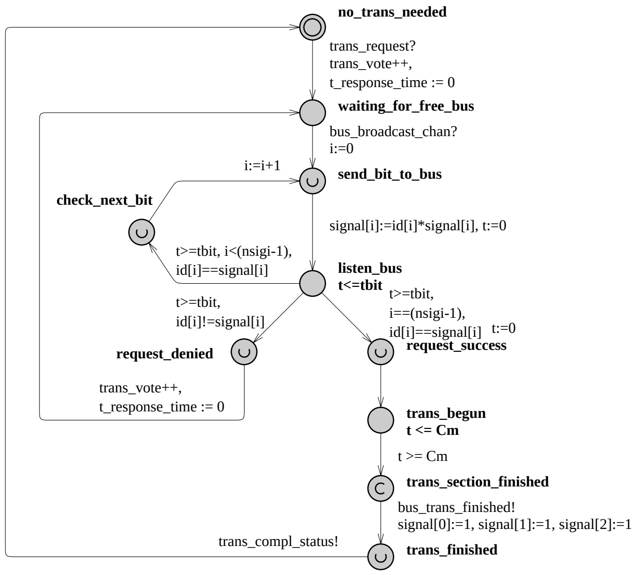


Fig. 4. Transceiver model

is willing to start the transmission. If this is the case than the global synchronization is realized via *bus_broadcast_chan* from the bus model.

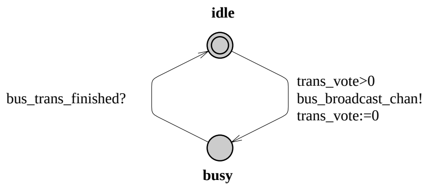


Fig. 5. Bus model

2.4 Application process model

As seen from Figure 6 the case study assumes 4 processors to be connected via CAN. Each processor is running one application process transmitting the messages of the same identifier. The application processes 1, 2, and 3 are periodic processes transmitting messages with identifier 1, 2, and 3 respectively. The application process 4 is a sporadic process transmitting the lowest priority message with identifier 4.

The periodic application process, with period T_m, is depicted in Figure 7. Afterwards each message is delayed by an operating system delay - the time between zero and J_i (called jitter in (Tindell and Burns, 1994)). Then the transmission request is done by *trans_request* channel. When the message is transmitted the process is informed by *trans_compl_status* channel. Location *no_transmission_activity* represents a situation when the process does not perform transmission, i.e. it performs for example computations. Location *init_location* starts the first task period, delayed by time between zero and T_m in order to represent the phase shift of the task.

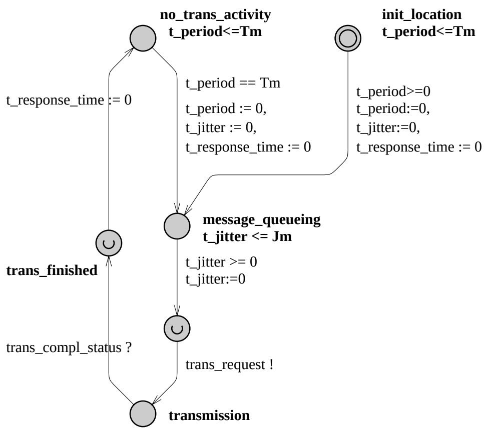


Fig. 7. Periodic application process model

The sporadic process model is depicted in Figure 8. Locations *no_trans_activity_*1 and *no_trans_activity_*2 represent a situation when the process does not perform any transmission. The process resides an arbitrary time in location *no_trans_activity_*1, then the transmission request is generated and when the message is transmitted the process is informed by *trans_compl_status* channel, and then the process has no influence on the bus. Local variable *t_response_time* in both models is used in properties to be verified.

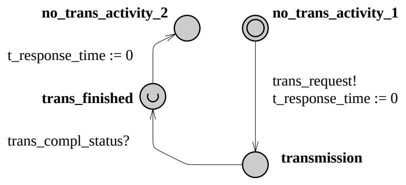


Fig. 8. Sporadic application process model

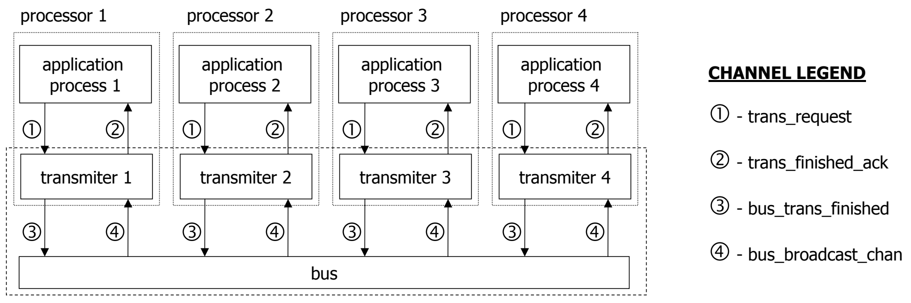


Fig. 6. Case study system configuration

3. VERIFICATION OF THE MODEL

The section presents the case study with periodic and sporadic processes including comparison with Tindell's approach (assuming 125kbps baudrate).

Table 1. Process parameters table

Msg.ID	Type	Period T_m [μsec]	$Deadline$ [μsec]	C_m [μsec]
1	periodic	2000	2000	504
2	periodic	3000	3000	504
3	periodic	5000	4000	504
4	sporadic	-	-	1040

Timing and logical properties to be verified can be for example the following ones:

(1) Is the system deadlock free?
(2) Is there any state in which processor 1 and processor 2 are in the data transmission section?
(3) Is there any situation in which the highest priority message does
not win the arbitration?
(4) Are all periodic messages transmitted prior to their deadlines?
(5) What is the worst-case response time R_m of the message with
identifier m (for m=1, 2 or 3)?

These properties are formulated in the temporal logic based formalism used in the UPPAAL verification tool UPPAAL (Pettersson and Larsen, 2000) as follows:

(1) A □ (not deadlock)
(2) E ◇ (Transceiver_1.request_success and Transceiver_2.request_success)
(3) E ◇ (Transceiver_1.request_denied)
(4) A □ (Process_m.trans_finished) ⇒ (Process_m.t_response_time < $Deadline$)
(5) A □ (Process_m.trans_finished) ⇒ (Process_m.t_response_time < R_m)

The verification results of timed automata tool are as follows:

(1) Property is satisfied
(2) Property is not satisfied
(3) Property is not satisfied
(4) See the section bellow
(5) R_m found by iteration (using bisection) see the section bellow

4. CASE STUDY

In this section we assume the configuration depicted in Figure 6 where each processor is running one application process transmitting one type of message (the message ID is equal to the application ID is equal to the processor ID). Table 1 shows parameters of three periodic and one sporadic process. The aim of the case study is twofold:

- to verify whether the response time satisfies a given deadline of the message (corresponding to property 4)
- to find the worst-case response time R_m iteratively by repeating the verification for different values of deadline (corresponding to property 5).

Table 2. Results of the experiment related to property 4 and 5

Message ID	$J_m[\mu sec]$	formula 4 result	$R_m[\mu sec]$
1	0	satisfied	1544
2	0	satisfied	2048
3	0	satisfied	3056
4	-	-	-

Table 2 shows verification results of the experiment related to property 4 and 5 without operating system delay . The response time of each periodic message is shorter then corresponding deadline assuming also relatively long sporadic message.

Table 3. Results of the experiment related to property 4 and 5 with operating system delay

Message ID	$J_m[\mu sec]$	formula 4 result	$R_m[\mu sec]$
1	456	satisfied	2000
2	0	satisfied	2552
3	0	satisfied	3056
4	-	-	-

Table 3 shows results of the experiment related to property 4 and 5 with operating system delay (J_m).

Values of R_m in tables 2, 3 are identical to those calculated by iterative algorithm (Tindell and Burns, 1994) based on equation

$$R_m = C_m + J_m + w_m$$

where

$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j$$

The term B_m presents the longest time that the given message m can be delayed by lower priority messages, the τ_{bit} is the bit time of the bus. The set $hp(m)$ is the set of messages of higher priority then message m.

5. CONCLUSIONS AND FUTURE WORK

This article shows a way of communication protocol modeling by timed automata. Model of entire distributed application can be obtained simply by interconnection with real-time operating system (Waszniowski and Hanzalek, 2003) and application software automata. The resulting model is suitable for verification of desired/undesired states in the time critical distributed applications like automotive IT based on CAN and OSEK. Due to this modular approach, the model is simply extensible to TT CAN.

Verification results of the CAN model can be directly compared to the results achieved by (Tindell and Burns, 1994), when obtaining identical results from both approaches (try to compute R_m for values in Tables 2 and 3). Moreover our approach can be simply extended to deal with internal logic structure of the application processes, processor sharing managed by operating system etc. On the other hand, high complexity is a drawback of the model checking approach in contrast to quite straightforward equations of the scheduling theory. For example verification of system deadlock, which is the most time consuming among tested properties, took 20 minutes for parameters in Table 1. on AMD-Athlon XP 1,8GHz computer with 1,3GB RAM.

Therefore our future work is related to hierarchical modelling and verification of distributed systems, i.e. parts of the system will be modelled and their timing parameters will be verified. Consequently the part will be replaced by a location with upper and lower bound on its timing parameters and it will be used for verification of upper layer. This approach certainly gives a little pessimistic result, but when tailored to the application logic, it can be rewarding.

Acknowledgements: This research work has been supported by the research program No. J04/98:212300013 "Decision Making and Control for Manufacturing" of the Czech Technical University in Prague and by IST project 35102 OCERA.

REFERENCES

Alur, Rajeev and David L. Dill (1994). A theory of timed automata. *Theoretical Computer Science* **126**(2), 183–235.

Berard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci and P. Schnoebelen (2001). *Systems and Software Verification: Model-Checking Techniques and Tools*. Vol. 7 of *ISBN: 3-540-41523-8*. Springer.

Buttazzo, G.C. (1997). *Hard Real-Time computing systems*. Kluwer Academic Publishers.

Corbett, James C. (1996). Timing analysis of Ada tasking programs. *IEEE Transactions on Software Engineering* **22**(7), 461–483.

Etschberger, K., Roman Hofmann, Joachim Stolberg, Christian Schlegel and Stefan Weiher (2001). *Controller Area Network: Basics, Protocols, Chips and Applications*. ISBN: 3-00-007376-0. IXXAT Automationpress.

Geischeder, Manfred, Klaus Gresser, Adam Jankowiak, Jochem Spohr, Andree Zahir, Markus Schwab, Erik Svenske, Maxim Tchervinsky, Ken Tindell, Gerhard Gser, Carsten Thierer, Winfried Janz and Volker Barthelmann (2000). Osek/vdx: Specification 2.1.

Katoen, Joost-Pieter (1999). Concepts, algorithms, and tools for model checking.

Klein, M.H., T. Ralya, B. Pollak, R. Obenza, M. Gonza and L. Harbour (1993). Practitioners handbook for real-time analysis: Guide to rate monotonic analysis for real time systems.

Liu, Jane W.S. (2000). *Real-Time Systems*. Prentice Hall.

Pettersson, Paul and Kim Guldstrand Larsen (2000). Uppaal2k.

Tindell, K. and A. Burns (1994). Guaranteed message latencies for distributed safety critical hard real-time networks.

Waszniowski, Libor and Zdenek Hanzalek (2003). Analysis of real-time operating system based applications. *FORMATS 2003*.