F2004F442

VERIFYING REAL-TIME PROPERTIES OF CAN BUS BY TIMED AUTOMATA

Krakora Jan*, Hanzalek Zdenek
Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering, Karlovo namesti 13, Prague 2, 121 35, Czech Republic

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

ABSTRACT
In the last years the in-car communications play more and more important role in automotive technology. With growing complexity of these systems, it is rather difficult to guarantee their correct behaviour. Therefore it is needed to apply new techniques for their analysis.

This article deals with verification of a typical automotive IT equipment based on a distributed system. Such system consists of an application SW (designed by application developer) 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.

Well known Rate Monotonic Scheduling (RMS) can be used to guarantee schedulability, when the application consisting of periodic processes is running on mono-processor with priority based pre-emptive kernel and the processes have their deadlines at the end of their periods. VOLCANO predicts the worst-case latencies for CAN as a direct application of the scheduling theory where the common bus is considered as shared resource. 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, considering their occurrence ratio.

This article presents an alternative approach based on model checking while using timed automata and temporal logics. 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; therefore the model complexity grows gradually at a design stage. Product of these automata is further used for checking of desired model properties. Important part of this article is CAN arbitration model composed of several timed automatons: bus automaton and transmitter automaton per each message ID.

Verification of the CAN model is compared to the results achieved by VOLCANO and it is enlarged to deal with internal structure of the application processes, operating system parameters in ABS case study. Moreover, while using the verification approach, one can verify not only the schedulability, but also rather complex properties linked to logic and timing behaviour of the distributed system. On the other hand, high complexity is a drawback of the model checking approach in contrast to quite straightforward equations of the scheduling theory.

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 (10)) while using several processors interconnected via standard broadcast communication based on the Controller Area Network (CAN) (8). Structure of the application under consideration is depicted in Fig. 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 (9)) or with the bus sharing (e.g. CAN message latency studied by Tindell (6)).

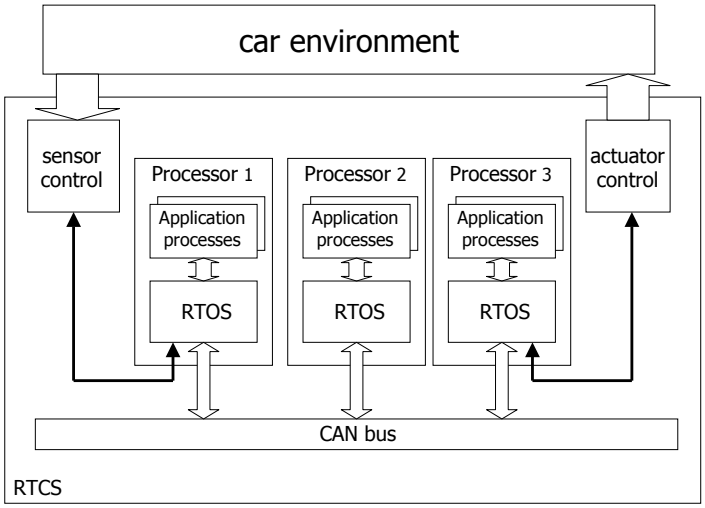

Fig. 1. Real time control system structure

The task schedulability on monoprocessor and multiprocessor systems is widely studied subject (11,12). For example Rate Monotonic Scheduling (RMS) can be used to guarantee schedulability, when the application consisting of periodic processes is running on mono-processor with priority based pre-emptive 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 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 (6). 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 (1) and temporal logics (2). 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 (5) on issues related to implementation and complexity of verification algorithms.

Modelling and verification of concurrent processes sharing one processor have been shown in (3, 4). These works incorporate priority based pre-emptive and non-pre-emptive scheduling, inter-task 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 (6) 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 behaviour 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 behaviour of CAN is modelled incorporating models of transceiver, bus and application process. Sections 3 provides verification of timing and logic properties and Section 4 compares this approach with Tindell's approach on example with periodic and sporadic processes. Section 5 shows a case study of ABS distributed real-time system.

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 (6) 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 checking 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 behaviour 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 (7) is shown in Fig. 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.

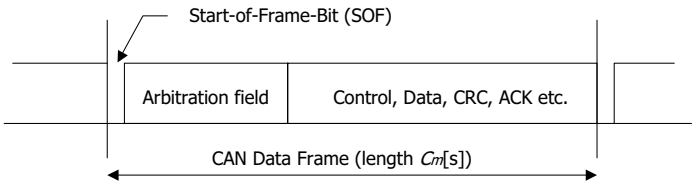

Fig. 2. CAN message frame format

After processing of the Start of Frame Bit (SOF) (see the CAN message frame format in Fig. 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 $t <= tbit$ in *listen_bus* location and guards $t>=tbit$, $t>=0$ 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 $t <= tbit\_u$ and guard $t>=tbit\_l$.

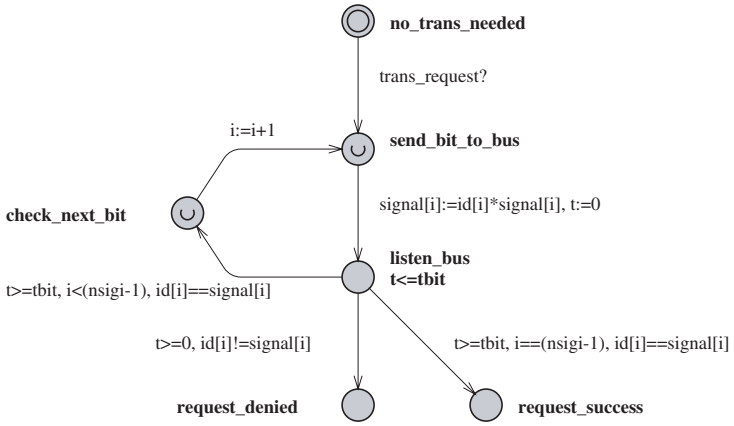

Fig. 3 Arbitration model (in UPPAAL like notation)

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 Fig. 4, and its interconnection with other automata is shown in Fig. 6. It is composed of the three sections:

- the arbitration section described already in Fig. 3
- 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
- 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 processor's 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).

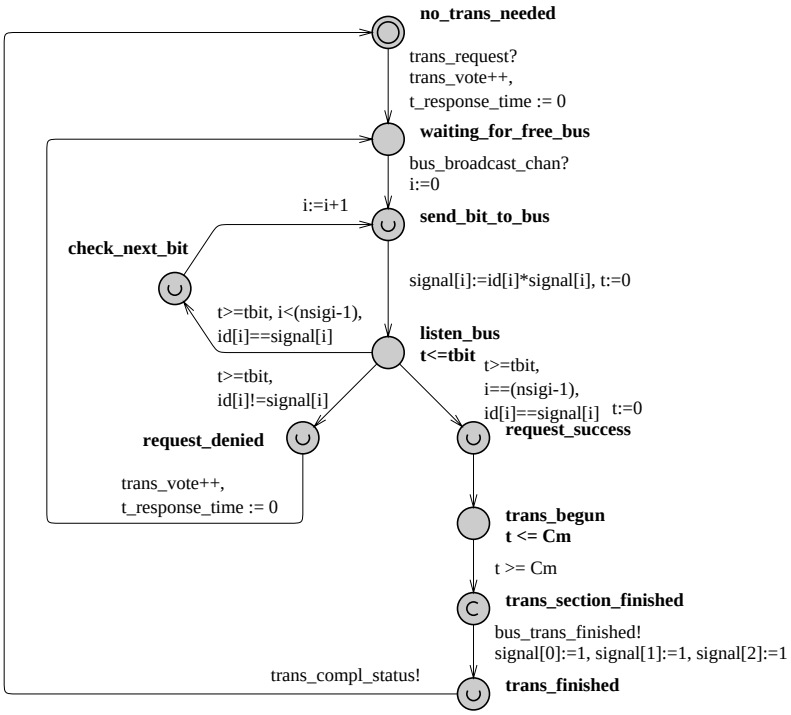

Fig. 4 Transceiver model

2.3 Bus Model

Fig. 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 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.

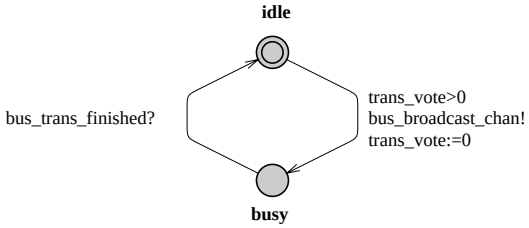

Fig. 5 Bus model

2.4 Application Process Model

As seen from Fig. 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.

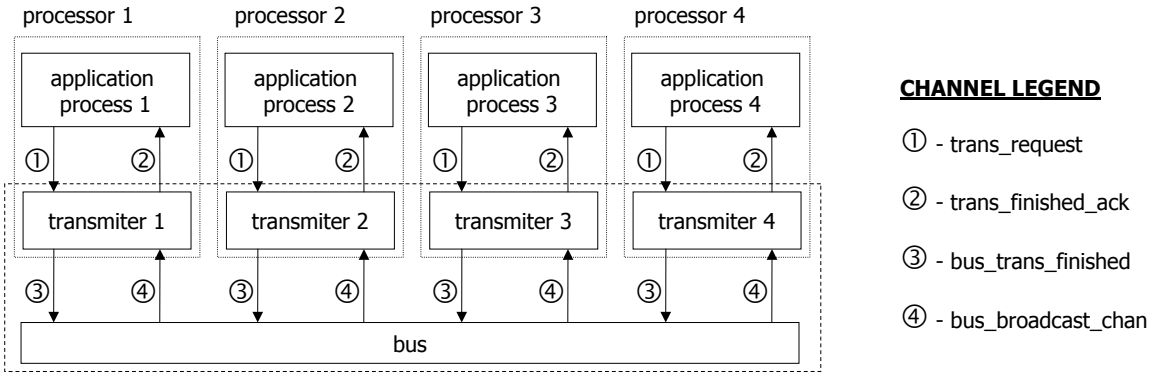

Fig. 6 Case study system configuration

The periodic application process, with period T_m, is depicted in Fig. 7. Afterwards each message is delayed by an operating system delay – the time between zero and J_i (called jitter in (6)). 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.

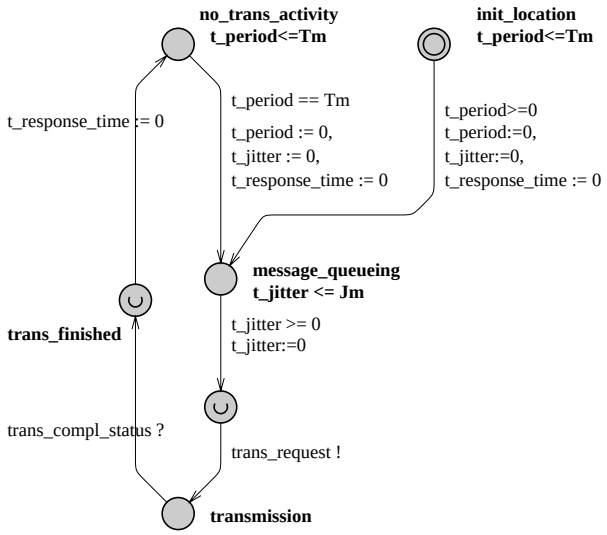

Fig. 7 Periodic application process model

The sporadic process model is depicted in Fig. 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.

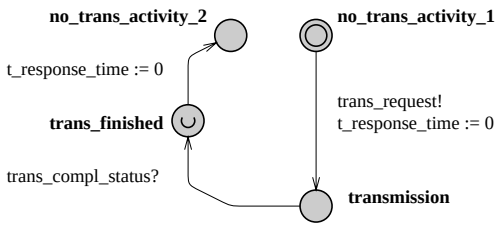

Fig. 8 Sporadic application process model

3. VERIFICATION OF THE MODEL

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

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 (7) 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. COMPARISON WITH TRADITIONAL APPROACH

In this section we assume the configuration depicted in Fig. 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).

Msg.*ID*	Type	Per. 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

Table 1: Process parameters table

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

Message *ID*	J_m	formula 4 result	R_m
1	0	Satisfied	1544
2	0	Satisfied	2048
3	0	Satisfied	3056
4	-	-	-

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

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

Message *ID*	J_m	formula 4 result	R_m
1	456	Satisfied	2000
2	0	Satisfied	2552
3	0	Satisfied	3056
4	-	-	-

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

Values of R_m in tables 2., 3. are identical to those calculated by iterative algorithm (6) based on equation

$$R_i = J_i + w_i + C_i$$

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. CASE STUDY

This case study is example of distributed system containing timed automata models, including the CAN model, the RT operating system model (3) and the Anti-lock Brake System (13) realised as application process model (see Fig. 9.). The system consists of two processors (i.e. MCUs) with pre-emptive RTOS (e.g. OSEK), communicating via CAN. The first processor (see Fig. 12.), connected to the brake pedal (see Fig. 10), detects the pedal position and transmits corresponding messages to the second processor. The second one (see Fig. 15) acquires information about acceleration/deceleration from an acceleration sensor, it receives messages from the first processor, and calculates and accomplishes a control action following rules of ABS.

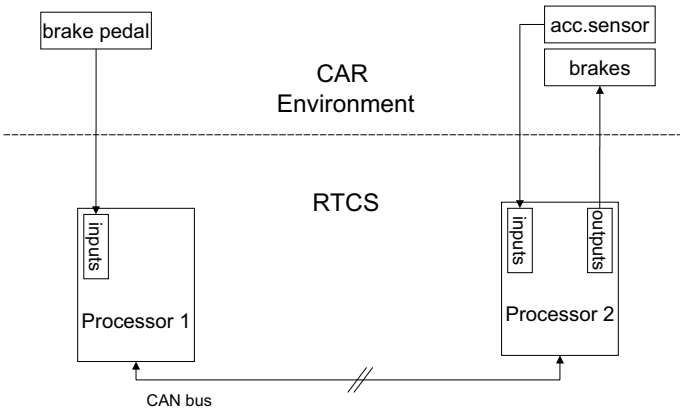

Fig. 9. Structure of the distributed system - ABS control

The timed automaton in Fig. 11 models typical situation - the pedal is pressed and then it is released after some undefined time. Model consists of three locations. Variable *BPPEvent* is set when the pedal is pressed (*bPedalInit* → *bPedalPressed*) and variable *BPREvent* is set when when it is released (*bPedalPressed* → *bPedalReleased*). Variable *BPPEvent* is read by read by a timed automaton model of *Task1* (see Fig. 13) and variable *BPREvent* is read by read by a timed automaton model of *Task2* (see Fig. 14).

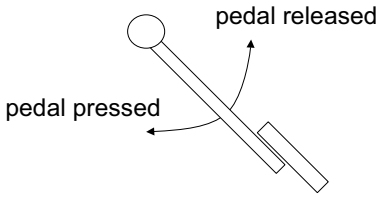

Fig. 10. Two state brake pedal

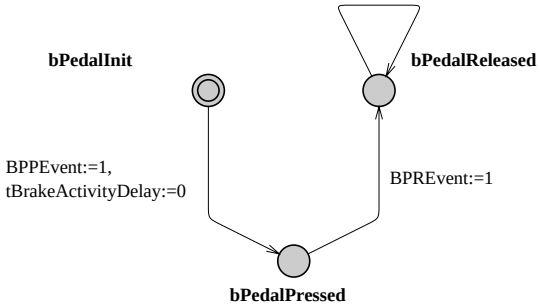

Fig. 11. Brake pedal model

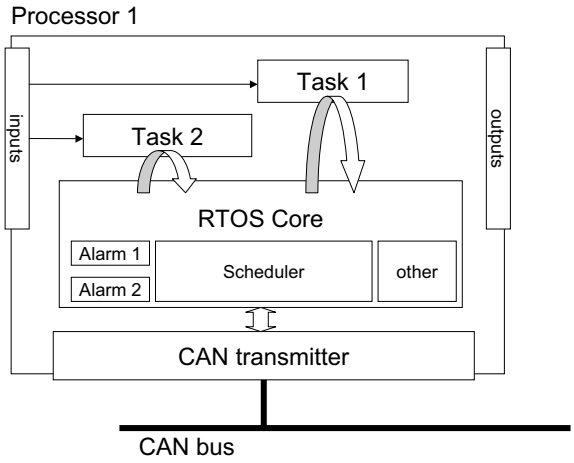

Fig. 12. Processor 1 structure

The model of the first processor is described in Fig. 12. The model consists of two automata, modelling the application tasks (*Task1*, *Task2*) periodically triggering the inputs, group of automata modelling RTOS including pre-emptive scheduler and periodic alarms (3), and group of automata modelling CAN as explained in previous sections. As shown below, the first task detects if the brake pedal is pressed and the second one if the pedal is released.

```
Task1RTOS1 () {
    // send message when the pedal is pressed
    if (BPPEvent) sendMsg(PedalPressed);
    // otherwise terminate the task - wait for the next activation
    TerminateTask();
};

Task2RTOS1 () {
    // send message when the pedal is released
    if (BPPEvent) sendMsg(PedalReleased);
    // otherwise terminate the task - wait for the next activation
    TerminateTask();
};
```

Timed automaton model of *Task1* is depicted in Fig. 13. *Task1* becomes ready (location *Ready1*) when it is triggered by an alarm (variable *nActivateBPPMT* and channel *wQuch*) and further it is executed (location *Comp1*) when it is the highest priority task in the OS queue (array *Q1*). *if* statement (location *Comp1*) has execution time bounded by its lower (constant

L1) and upper bound (constant *U1*). When *BPPevent* is not set then the task terminates. Otherwise *sendMsg* function (locations *Waiting*, *Ready2*, *Comp2*), with execution time bounded by its lower (constant *L2*) and upper bound (constant *U2*), is executed. Similarly the timed automaton model of *Task2* is depicted in Fig. 14.

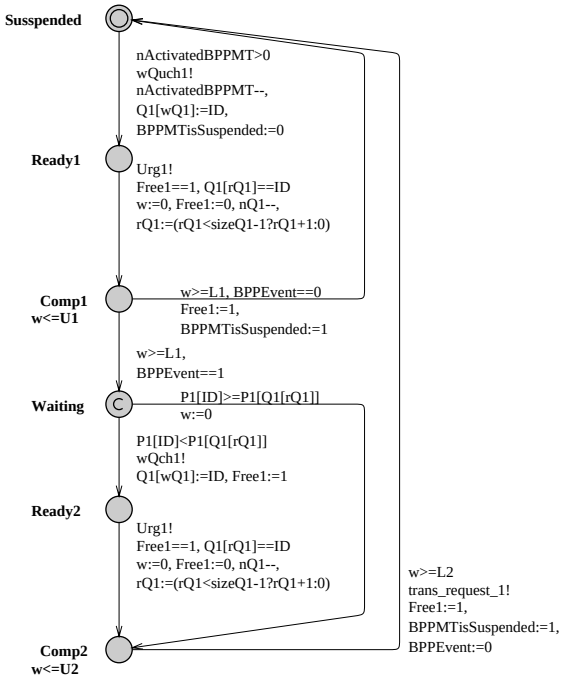

Fig. 13. Timed automaton model of *Task1*

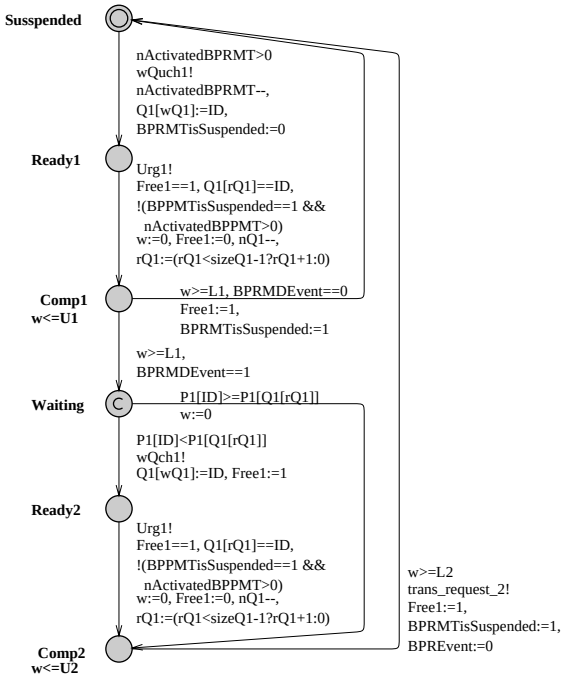

Fig. 14. Timed automaton model of *Task2*

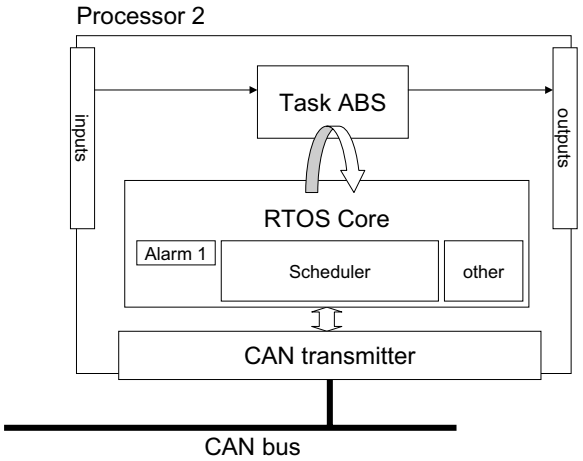

Fig. 15. Processor 2 structure

The second processor, depicted in the Fig. 15, includes *TaskABS* which receives the messages (the pedal position), it reads local input (acceleration sensor), it calculates ABS controller and accomplishes a control action (brake shoes). When braking (variable *BPPMEvent ==1* in Fig. 16), the ABS controller is looking for decelerations in the wheel that are out of the ordinary (guard *acceleration <= MAXdec*). Right before wheel locks up, it will experience a rapid deceleration. If left unchecked, the wheel would stop much more quickly than any car could. The ABS controller knows that such a rapid deceleration is impossible, so it reduces the

pressure to that brake (location *brake_shoes_released*) until it sees an acceleration (guard *acceleration > 0*), then it increases the pressure until it sees the deceleration again. It can do this very quickly, before the tire can actually significantly change speed. The result is that the tire slows down at the same rate as the car, with the brakes keeping the tires very near the point at which they will start to lock up. Corresponding detailed models (in UPPAAL notation) of *ABSTask*, brake shoes and acceleration sensor are depicted in Fig. 17., Fig. 18. and Fig. 19.

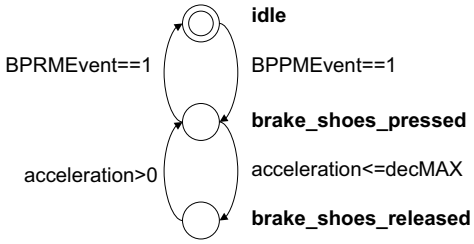

Fig. 16. ABS controller state diagram

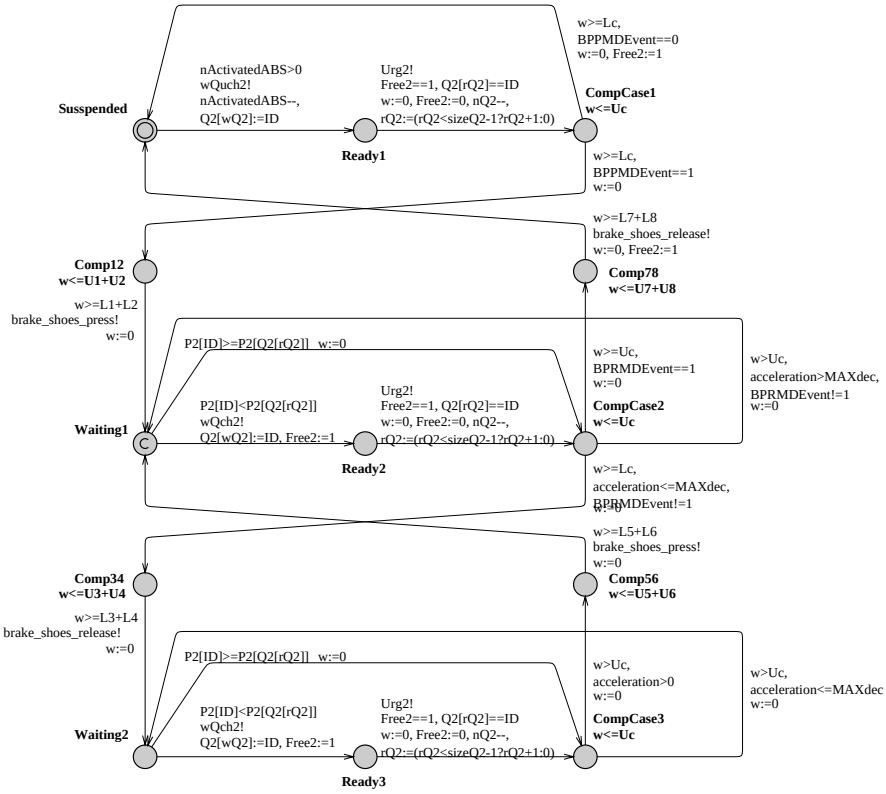

Fig. 17. Timed automaton model of *ABSTask* algorithm

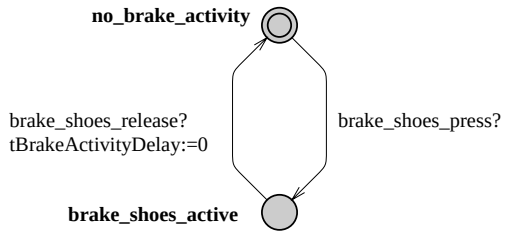

Fig. 18. Timed automaton model of *brakes*

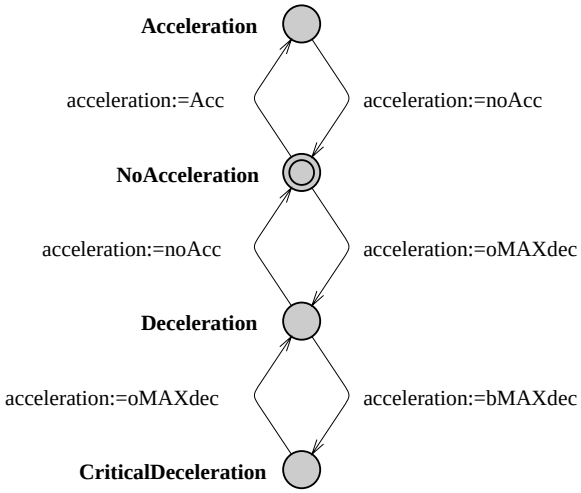

Fig. 19. Timed automaton model of *acceleration sensor*

5.1 Verification

The system parameters are shown in Table 4, Table 5 and Table 6 for this case study.

Task name	Task period [μsec]	*U1,U2,L1,L2* [μsec]
Task1	5000	1
Task2	5000	1

Table 4: Processor 1 RTOS system parameters

Task name	Task period [μsec]	*U1...U8, L1...L8* [μsec]
TaskABS	5000	1

Table 5: Processor 2 RTOS system parameters

Message *ID*	C_m [μsec]	bit time [μsec]
1	504	8
2	504	8

Table 6: Message Parameters

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

1. Is the system deadlock free?
2. When the pedal was pressed and not released, the *PedalReleased* message would not be received.
3. Message *PedalPressed* is received at least 2ms after the pedal has been pressed.
4. What is the worst case receive time for message *PedalPressed*?
5. Will be ever the ABS active?
6. What is the worst-case time for activation of brake shoes?

These properties are formulated in the temporal logic based formalism used in the UPPAAL verification tool UPPAAL (7) as follows:

1. A□ not deadlock
2. (BrakePedal.bPedalPressed and not BrakePedal.bPedalStillReleased) --> (not BPRMDEvent==1)
3. A□ (tBrakeActivityDelay>2000 and not BrakePedal.bPedalReleased) imply (BPPMDEvent==1)
4. A□ (tBrakeActivityDelay>*X* and not BrakePedal.bPedalReleased) imply (BPPMDEvent==1)
5. E◊ (R2T1.Sheduled2)
6. A□ (Brake.brake_shoes_active and not BrakePedal.bPedalReleased and not BrakePedal.bPedalStillReleased) imply (tBrakeActivityDelay<*X*)

The verification results of timed automata tool are as follows:

1. Property is satisfied
2. Property is satisfied
3. Property is not satisfied
4. *X*=5507 - found by iteration (using bisection)
5. Property is satisfied
6. *X*=10007 - found by iteration (using bisection)

6. CONCLUSION AND FUTURE WORK

This article shows a way of communication protocol modelling by timed automata. Model of entire distributed application can be obtained simply by interconnection with real-time operation system (3) 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 e.g. to TT CAN.

Verification results of the CAN model can be directly compared to the results achieved by (6), when obtaining identical results from both approaches (try to compute R_m for values in Tables 2,3). 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 1,8 GHz computer with 1,3 GB 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, then it can be quite practical.

7. REFERENCE

(1) Rajeev Alur and David L. Dill, *"A theory of timed automata"*, Theoretical Computer Science (vol 126) pages: 183—235, 1994

(2) Joost-Pieter Katoen, *"Concepts, Algorithms, and Tools for Model Checking"*, IMMD, 1999

(3) Libor Waszniowski and Zdenek Hanzalek, *"Analysis of Real-Time Operating System Based Applications"*, FORMATS 2003, 2003

(4) James C. Corbett, *"Timing Analysis of {A}da Tasking Programs, IEEE Transactions on Software Engineering",* (vol 22) pages: 461—483, 1996

(5) B. Berard and M. Bidoit and A. Finkel and F. Laroussinie and A. Petit and L. Petrucci and P. Schnoebelen, *"Systems and Software Verification: Model-Checking Techniques and Tools",* (vol 7), Springer, 2001

(6) K. Tindell and A. Burns, *"Guaranteed Message Latencies for Distributed Safety Critical Hard Real-Time Networks"*, 1994

(7) Paul Pettersson and Kim Guldstrand Larsen, *"UPPAAL2k"*, 2000

(8) K. Etschberger and Roman Hofmann and Joachim Stolberg and Christian Schlegel and Stefan Weiher, *"Controller Area Network: Basics, Protocols, Chips and Applications"*, IXXAT Automationpress, 2001

(9) M.H. Klein and T. Ralya and B. Pollak and R. Obenza and M. Gonza and L. Harbour, *"Practitioners Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real Time Systems"*, (vol), Kluwer Academic Publishers, 1993

(10) Manfred Geischeder and Klaus Gresser and Adam Jankowiak and Jochem Spohr and Andree Zahir and Markus Schwab and Erik Svenske and Maxim Tchervinsky and Ken Tindell and Gerhard Göser and Carsten Thierer and Winfried Janz and Volker Barthelmann, *OSEK/VDX: Specification 2.1*, 2000

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

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

(13) Nice Kerim, *"How Anti lock Brakes works"*, http://www.howstuffworks.com