Timed Automata Approach to Real Time Distributed System Verification

Jan Krákora, Libor Waszniowski, Pavel Píša and Zdeněk Hanzálek
Czech Technical University in Prague,
Department of Control Engineering, Centre for Applied Cybernetics
Karlovo namesti 13, Prague 2, 121 35, Czech Republic
{krakorj,xwasznio,pisa,hanzalek}@fel.cvut.cz

Abstract

This article deals with a distributed real-time application modelling by timed automata. The application under consideration consists of several processors communicating via Controller Area Network (CAN); each processor executes an application that consists of tasks running under an operating system (e.g. OSEK) and using inter-task synchronization primitives. For such system, model checking algorithm implemented in a model checking tool (e.g. UPAALL) can be used to verify complex time and logical properties of the proposed model (e.g. end-to-end response time, state reachability, deadlock freeness).

Since the proposed timed automata model contains more crucial details of the system behavior with respect to classical approaches to the response time analysis, the model checking approach provides less pessimistic results in many cases.

Keywords: *distributed Real-Time System, Controller Area Network, Timed Automata, Model Checking*

1 Introduction

Let us assume a distributed real time control system consisting of application tasks running under an Operating System (OS e.g. OSEK [9]) while using several processors interconnected via Controller Area Network (CAN) [4]. The structure of the application under consideration is depicted in Figure 1. The crucial problem is to verify both, time properties (e.g. end-to-end response time, schedulability of periodic processes, response time) and logic properties (e.g. deadlock freeness, mutual exclusion) of the applications incorporating two kinds of shared resources - the processor and the bus.

Incorporation of both kinds of resources have been solved in [10, 7]. These methods do not consider the internal structure of the tasks (e.g. data dependent branching and loops) and the model of the environment affecting release times of the tasks and messages. Scheduling theory based methods therefore compute the task finishing time by summing the task's worst-case execution time, the duration of the preemption by higher-priority tasks in the worst-case inter-arrival times and phasing of higher priority tasks, and the worst-case blocking by lower-priority tasks on shared resources. Similarly, the message worst-case delivery time is computed by adding together its worst-case length, the maximal length of one lower priority message (since a high priority message cannot interrupt the message that is already transmitted) and due to the priority based bus arbitration, the duration of the transmission all higher priority messages in the worst-case occurrence ratio and phasing. The worst-case response time is a conservative abstraction of all possible situations but it could be too pessimistic in many applications, since all the mentioned worst cases do not occur at the same time (see examples in [11, 3]).

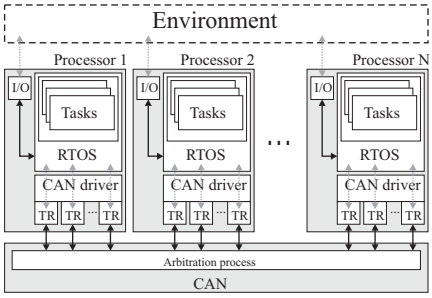

Figure 1. Distributed Real-Time System

This article presents an alternative approach based on the model checking method [2] while using the timed automata model [1]. Using this approach we model parts of the distributed system (application tasks, operating system and communication bus) and the controlled environment (parts affecting the releasing of tasks and messages) by timed automata and then using model checking tool UPPAAL [8] to verify the model's functional and time properties formalized by a subset of timed computation tree logic (TCTL see [2, 8]).

Our approach assumes a *fine grain model* treating the task internal structure, OS kernel, communication layer and the controlled environment. The task model consists of several blocks of code called *computations* (given by $BCET$ and $WCET$), calls of OS services, selected variables, code branching and loops (affected by values of se-

lected variables). The OS kernel model formalizes operating system services functionality. The communication layer model describes messages arbitration and transmission length. The controled environment model, specifying arrival times of events, releasing the tasks or the messages, plays the key role in the system verification. An exhaustive analysis of this model behavior (automatically done by model checking tool UPPAAL) considers task and messages response times corresponding to a realistic phasing, realistic tasks and messages arrival times, realistic blocking and realistic execution time in a relation to the modeled code branching. Therefore the verification result is as precise as the model is precise.

Moreover, while using the model checking approach, one can verify not only end-to-end response time (i.e. the time of the actuator reaction on the sensor), but also rather complex properties linked to logic and timing behavior of the distributed system (e.g. deadlock, state reachability, safety and bounded liveness properties). On the other hand the complexity is a drawback of the model checking approach in contrast to the straightforward equations of the scheduling theory.

The paper is organized as follows. Section 2 describes a CAN timed automata model consisting of a transceiver and a bus arbitration. Sections 3 shows a simple example of the application task model and Section 4 briefly describes the OS kernel model. Section 5 demonstrates on a simple case advantages of the model checking approach.

2 CAN timed automata model

CAN models, based on discrete event system, have been shown in [5, 6]. This section deals with the modelling of CAN [4] by timed automata [8], focusing on the reduction of the verification process complexity. The model consists of one arbitration automaton and one transceiver automaton per each message transmited in the system. The Interface between the transceiver and the upper layer model (tasks) is provided by synchronization channels *SendMsg* and *ReceiveMsg*.

2.1 Transceiver automaton

One transceiver automaton for message *msgID* is depicted in Figure 2. Location *no_trans_request* represents the situation when the upper layers (tasks etc.) do not request sending any data. Request to send a data is modelled by *SendMsg[msgID]* channel. Location *msg_queuing_jitter* represents the delay of a message given by e.g. a driver. The timed automaton spends time from 0 to *Jm* in this location. After this time the transceiver requests an access to the bus. It is modeled by the synchronization with the arbitration automaton via channel *transmit* and by setting *msgID* item in *arb_participant* array. The transceiver is allowed to access the bus via *arb_success[msgID]* channel. The time of the data transmission is given by a value from *CmL* to *CmU*. When the message is transmited the model informs the arbiter via *arb_ack[msgID]* channel and resets the *arb_participant[msgID]* item. Transceiver also informs the upper layer by *ReceiveMsg[msgID]* and waits for the next request.

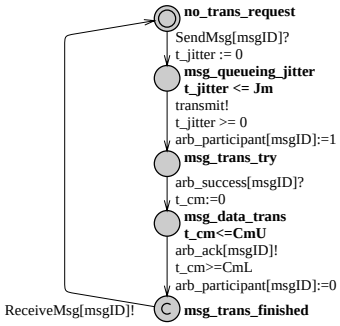

Figure 2. Transceiver automaton (in UPPAAL notation)

2.2 Arbitration automaton

Access to the CAN bus is based on a non-destructive bite-wise arbitration mechanism modeled by the timed automaton depicted in Figure 3.

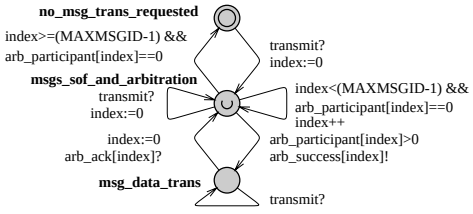

Figure 3. Arbitration automaton

In location *no_msg_trans_requested* the automaton waits till any transceiver automaton requests to transmit its message (by channel *transmit*). In location *msgs_sof_and_arbitration* the highest-priority requesting message to be transmited is chosen. The transceiver of the highest-priority message is informed by *arb_success[index]* channel and the arbitration automaton waits in *msg_data_trans* till the end of the transmission. When the transmission is finished (*arb_ack[index]* channel) either a new requesting message is chosen in *msgs_sof_and_arbitration* location or the automaton returns to location *no_msg_trans_requested*.

The arbitration mechanism is modeled (in location *msgs_sof_and_arbitration*) as follows. The *arb_participant* array is read item by item (from 0 to the highest index i.e. from the highest to the lowest priority). If its value is different from zero, the message with $msgID = index$ wins the arbitration process. The corresponding transceiver automaton is informed via *arb_success[index]* channel.

3 Timed automata model of task

This chapter deals with a model of an application running under an operating system OSEK. Each task of the application is modeled by one timed automaton. Figure 4 demonstrates an example of a simple task executing *computation* $CompPID$, calling OS services $GetResource$, $ReleaseResource$ and $TerminateTask$ and sending messages $TorqueMsg$ and $BreakMsg$ via the CAN bus.

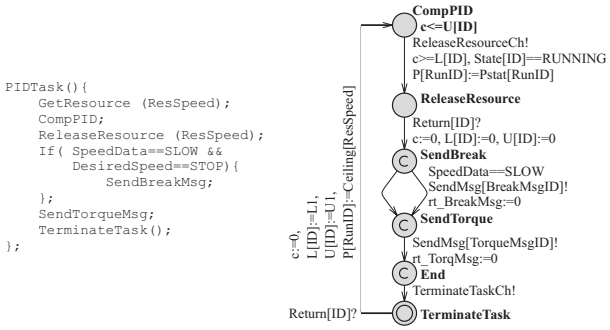

Figure 4. Simple task example

In the timed automaton, each *computation* is represented by one location of the same name. Time, spent in this location (measured by clock c), represents the *computation* response time (including preemption) and is bounded by the values stored in integers $L[ID]$ and $U[ID]$ (where index ID is unique tasks identifier). OS services calls are modeled by the transitions, synchronized via the channel of the corresponding name ($ReleaseResourceCh!$) and by the location ($ReleaseResource$) where the task waits to return from the service ($Return[ID]?$). The task priority is modified when the resource is acquired ($P[RunID] := Ceiling[ResSpeed]$) and released ($P[RunID] := Pstat[RunID]$; see [9]). Statement *if* is modeled by the automaton branching.

4 OS kernel model

The OS kernel model consists of timed automata representing the OS services functionality, timed automata managing the ready queue and the preemption, and of some variables representing OS objects. Each automaton, representing an OS service waits in its initial state until its function is called. Then it manipulates task states, the ready queue and other operating system objects (e.g. events), and chooses the highest-priority task to run. Only very simple OS services, e.g. $GetResource$, are not modeled by a particular automaton, but only by an assignment connected with a transition of the task automaton ($P[RunID] := Ceiling[ResSpeed]$).

It should be noted that the model is over approximate due to the preemption. It means that except the real-system behavior, it model also some additional non-determinism. But the task's worst-case response times are not affected by this non-determinism.

5 Case Study

This chapter presents main advantages of the model checking approach on a simple crane control system. The system consists of one node used to measure the rotation speed of the rope-drum, one node linked to the motor driving the rope-drum, one node controlling the locking break and one node with the control unit. All these nodes are interconnected via CAN (see Figure 5).

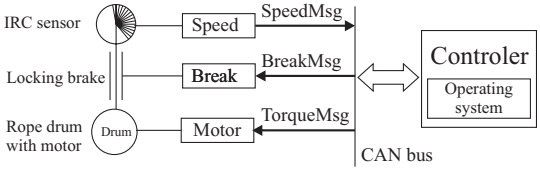

Figure 5. System configuration

The information about the speed is periodically transmitted as the highest-priority message *SpeedMsg* with the period varying according to the rope-drum rotation speed. The period is 200 time units when the speed is low, 150 when the speed is medium and 60 when the speed is high. The locking break is activated by medium priority message *BreakMsg* which is not sent more often than once per 150 time units. This message is sent only when the rope-drum rotation speed is low. The torque of the motor is updated by the lowest priority message *TorqueMsg* once per 150 time units.

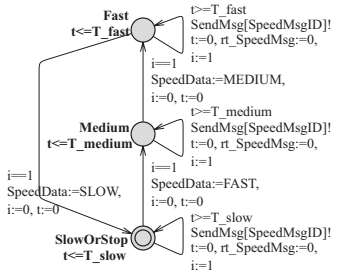

Figure 6. Rope drum automaton

The control unit executes two tasks. The highest-priority task *SpeedTask*, activated by the interrupt service routine when message *SpeedMsg* arrives, calculates the rotation speed (Figure 7). The lowest priority task *PIDTask*, periodically activated each 150 time units, generates two outputs - *TorqueMsg* and *BreakMsg* (Figure 4).

The model of the controlled system consists of timed automaton *Rope-Drum* (Figure 6) and timed automata *Motor* and *Break* (both are neglected here - they simply wait in a loop on *TorqueMsg* or *BreakMsg*, respectively). The model of CAN consists of timed automaton *Arbitration* (Figure 3) and three timed automata *Tranceiver* (one for each message). The Model of the control

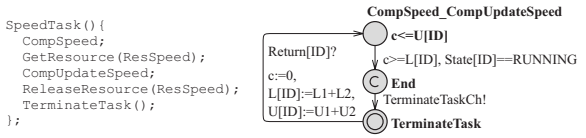

Figure 7. Speed task model

Table 1. Messages responce times by Model Checking approach

M_{name}	M_{ID}	T_f	T_m	T_s	L	R_{MC}
SpeedMsg	0	60	150	200	30	**60**
BreakMsg	1	–	150	150	30	**90**
TorqueMsg	2	150	150	150	30	**90**

M_{name} - message name; M_{ID} - message ID; T_f, T_m and T_s - message period during fast, medium and slow drum speed; L - message length; R_{MC} - Messages worst-case response time by model-checking approach (from queuing till delivering)

unit application software consists of timed automaton *SpeedTask* (Figure 7) and timed automaton *PIDTask* (Figure 4). Moreover, there are timed automata *CANisr* (it activates task *SpeedTask* by OS service $ActivateTask$, when it receives message *SpeedMsg*) and *AlarmPID* (it periodically activates task *PIDTask*). Both these automata are neglected here. The model of the operating system consists of the timed automata modelling the OS services (i.e. *ActivateTask*, *TerminateTask*, *ReleaseResource*) and the automaton modeling preemption not described here. The whole case study model can be downloaded from http://dce.felk.cvut.cz/cak/Research/RTVerif/RTVerif.htm.

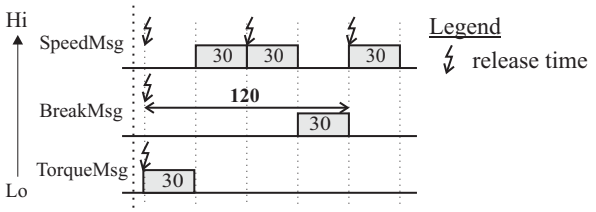

Figure 8. *BreakMsg* **Worst-Case schedule**

The described model can be used to verify logical or time properties (e.g. end-to-end response, state reachability). Table 1 shows the verified message response times (from queuing till delivering). Notice that the scheduling theory based analysis methods [10, 7] do not consider the logical aspect of the system behavior (in our approach modeled by timed automata *Rope-Drum* in Figure 5 and *PIDTask* in Figure 4). These methods therefore can not use the information that message *BreakMsg* can not be transmitted when the highest-priority message *SpeedMsg* is transmitted with the worst case period (60 time units). The scheduling-theory-based analysis methods must therefore analyze the worst-case schedule (depicted in Figure 8) that is pessimistic.

For this trivial case study, it would be possible to use the scheduling-theory-based analysis methods and analyze separately both situations - when message *BreakMsg* is transmitted and when not. However it is difficult or even impossible for systems with a more complex behavior.

6 Conclusion

The modeling method, presented in this article, applies to distributed applications running under OSEK on several processors, communicating via CAN.

Existing approaches for distributed real-time applications analysis (e.g. [7, 10]) use a very elegant way of deciding whether the application is schedulable. Opposite to these methods, the proposed model-checking approach based on the timed-automata model is less pessimistic, as presented in the case study, since it is based on a more detailed model including also the logic properties of the system and environment. Moreover, this approach provides a room to verify more complex properties (e.g. deadlock freeness, safety and bounded liveness properties).

On the other hand, the high complexity is a drawback of the model checking approach in contrast to quite straightforward equations of the scheduling theory. Currently, we are able to verify just small system configurations (i.e. 2 processors and 3 transmitted messages).

Acknowledgements: This work has been supported by project LN00B096 of Ministry of Education of the Czech Republic.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

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

[3] A. W. C.M. Bailey C.M., A. Burns and C. Forsyth. A performance analysis of a hard real-time system. *Pergamon*, 3(4):447–464, 1995. Control Engineering Practice.

[4] K. Etschberger, R. Hofmann, J. Stolberg, C. Schlegel, and S. Weiher. *Controller Area Network: Basics, Protocols, Chips and Applications*. ISBN: 3-00-007376-0. IXXAT Automationpress, 2001.

[5] G. Juanole. Modélis. Éval. protocol MAC du rseau CAN. *École ETR99*, Sept. 1999. Control Engineering Practice.

[6] J. Krákora and Z. Hanzálek. Timed automata approach to CAN verification. *INCOM 2004*, 2004.

[7] J. C. Palencia and M. G. Harbor. Schedulability analysis for tasks with static and dynamic offsets. *IEEE Real-Time Systems*, 1998.

[8] P. Pettersson and K. G. Larsen. *UPPAAL2k*, 2000. citeseer.nj.nec.com/pettersson00uppaalk.html.

[9] OSEK/VDX: Specification 2.1, 2000. http://www.osek-vdx.org.

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

[11] L. Waszniowski and Z. Hanzálek. Analysis of real-time operating system based applications. *FORMATS 2003*, 2003.