Asian Journal of Information Technology 5 (7): 706-711, 2006

Formal Modeling and Analysis of A Node Reintegration in The Time-Triggered Architecture

Aliouat Zibouda
Department of Engineer Science, Faculty of Computer Science,
Ferhat Abbes University Sétif, Algeria

Abstract: Time-Triggered Architecture (TTA) is distributed computer architecture for highly dependable real-time systems. The communication between nodes of this architecture is achieved by a key module that implements a Time-Triggered Communication Protocol (TTC/P). TTP/C integrates a set of fault-tolerant services like : message transmissions, clocks synchronization and Group Membership Protocol (GMP). The complexity and criticity of GMP and its sensitive interaction with other system components has led to the development of many versions. These different versions of GMP included more formalism in its analysis and verification but none of them, in our best knowledge, has dealt with the problem of node reintegration after recovery. We report the first formal modelling of a reintegration for a safety-critical distributed embedded system. A reintegration node increases system survivability by allowing a transiently-faulty node to regain a group. The group membership algorithm is formally specified by a set of guarded commands.

Key words: Time-Triggered Architecture, TTP/C, fault-tolerant algorithm, reintegration

INTRODUCTION

The Time-Triggered Architecture (TTA)[1,2] is distributed computer architecture for the implementation of highly dependable real-time systems. In particular, it targets on embedded control applications, such as by-wire systems in the automotive or aerospace industr[3]. For these safety-critical systems fault tolerance is of utmost importance. Embedded systems in this kind typically involve distributed computations and requirements of faults-tolerance. This specificity makes their analysis and verification of they behave as required inherently difficult. To obtain the kind of reliability required for highly safety-critical applications, mere testing alone is usually insufficient. In this regard, formal analysis can provide an additional source of confidence; it has argued by Rushby[4] that the necessary level of confidence in correct behaviour cannot be achieved without a careful formal analysis of the mechanisms and algorithms involved .

The Time-Triggered Protocol TTP/C[5-7] constitutes the core of the communication level of the Time-Triggered Architecture. It furnishes a number of important services, such as atomic broadcast, consistent membership and protection against faulty nodes, which facilitate the development of these kinds of fault-tolerant real-time applications. However, these protocol mechanisms rely on a rather optimistic fault hypothesis and assume that a fault is either a reception fault or a consistent send fault of some node[8].

Several aspects of TTP/C and related protocols have therefore been formally modeled and analyzed, including clock synchronization[9], group membership[10] and startup procedure[11]. A detailed overview of formal analysis work for the Time-Triggered Architecture is given by Rushby[12]. While so far the protocol algorithms of the time-triggered protocol have been the focus of the formal analyses cited above, we concentrate in this paper on the communication properties of TTP/C, thereby complementing and extending previous work[10].

The goal of this work is to formally model a group membership protocol dealing with node reintegration after recovery operation from failed computations, in the TTP/C group membership protocol.

THE TTA ARCHITECTURE

This architecture of distributed processors is destined to the execution of highly dependable systems in real time environment. TTA[13] is an embedded architecture composed of one or several clusters. A cluster is a set of independent nodes inter connected by a duplicated communication network providing broadcast transmission. Every node is divided in two subsystems: The Host subsystem and communication control subsystem (Fig. 1).

- The host level executes distributed real time applications.

Corresponding Author: Aliouat Zibouda, Department of Engineer Science, Faculty of Computer Science, Ferhat Abbas University Sétif, Algeria

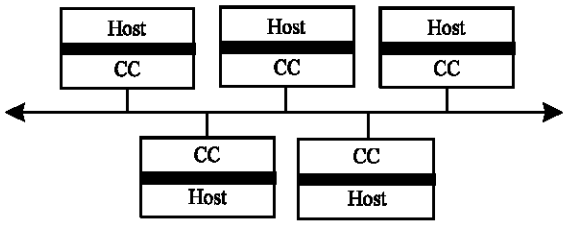

Fig. 1: Structure of a TTA cluster

- The communication control level is dedicated to the transmission of messages on the bus based on the TTP/C protocol[6].

The communication between these two levels takes place by the CNI module (Communication Network Interface). The access to the bus is controlled for every node by bus guardian. The use of a bus guardian enables suppressing the possible temporal fault in sending. Thus, a node won't be able to send messages outside the period of broadcast that is assigned to it. Indeed, having an independent temporal source and the knowledge of broadcast instants of every node in the MEDL (MEssage Descriptor List), the bus guardian allows access to the bus just during a limited length (window of broadcast). This strategy is suitable to solve the particular problem of the babbling idiot[14]. This problem characterizes faults in distributed systems where a node monopolizes the access to a resource (here the bus) creating so, the well known situation of starvation.

The TTP/C protocol : The TTP/C protocol implements communication between nodes by broadcast that takes place according to a TDMA (Time Division Multiple Access) access scheme, Fig. 2. Thus, every node is assigned a fixed and periodic duration of broadcast: The slot, during which it can send messages. All slots are regrouped in TDMA rounds; during a round, broadcast rights are granted once and one alone to each of nodes in a predefined order. It is from the MEDL that every node knows the broadcast instant of other nodes. To be able to broadcast at the predefined instant, each of nodes must synchronize its clock with the global clock. For this reason, a distributed algorithm of clock synchronization is executed[9].

GMP and faults hypothesis: In a distributed system, an adherence protocol of GMP group is a fault tolerant mechanism enabling to get a consensus on the identities of non failed (correct) processors. Any failed processor must be excluded from the group at the end of limited time. The studied protocol is based on the following hypothesis.

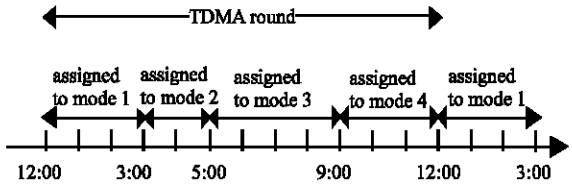

Fig. 2: TDMA round strategy

The system is composed of N processors, numbered from 0 to N-1, connected to a bus. The access to the bus is managed by a static organization scheme[10] according to which is assigned, to every processor, time intervals (slots) during which it can transmit. The other processors being then in receive state: At the slot t, only the processor t modulo N can transmit. However, it remains silent if it diagnosed itself as failed processor (i.e., it considers itself that it is no more in the group). In the contrary case, it sends a message including its local view of the group (i.e., its local view is the list of processors considered as non faulty, or membership).

Because of the presence of other fault tolerance mechanisms in TTP/C module, the GMP sub-module assumes that all fault occurrences may be only of two types:

- Send fault: These faults are supposed to be consistent that is, no processor of the group receives anything, or all receive something that is interpreted like an invalid message (invalid frame). In other words, no faults are generated by the communication bus.
- Receive fault. A processor affected by the type of this fault can't receive anything, or receives an invalid message.

From the moment where a processor becomes faulty (first fault manifestation), its behavior towards sending or receiving messages can be arbitrary (for instance, case of permanent or transient receive fault, send fault followed by receive one, etc). We assume that the faults occurrence is sufficiently rare to guarantee that when a processor fails, it flows out an interval of time greater than 2n slots before another processor of the group becomes faulty. Furthermore it will always remain at least two non faulty processors in the system. Let's note that this hypothesis of rarity of fault occurrences is based on the existing system experience[14].

Under the previous hypothesis the GMP protocol must guarantee the following properties at all times:

Validity of the local views of the group: At all times, non-faulty processors should have all and only the non-faulty processors in their membership sets, while faulty processors should have removed themselves from their sets. This requirement is, however impossible to satisfy as it may take some time to diagnose the faultiness of a processor. We therefore must allow a single faulty processor to be included in the membership set of non-faulty processors, while faulty processors may have (a subset of) the non-faulty processors plus themselves in their sets.

Agreement on members of the group: At all times all non-faulty processors should have the same membership sets.

Diagnosis in limited time: A processor that becomes faulty should eventually diagnose its fault and remove itself from its own membership set.

Algorithm description: In our model we assume a set proc of n processors, labeled 0, 1,..., n-1, that are arranged in a logical ring. Every processor p maintains a set mem^t_p (the membership set of processor p) that contains all processors that p considers operational at time t. In slot t the processor with label t mod n is the broadcaster, denoted broadcaster(t). In addition to the message data, the broadcaster sends those parts of its internal state that are critical for the protocol to work properly. More precisely, a CRC checksum that is calculated over the message data and the critical state information (which includes the membership set) is appended to the message. For the analysis of the group membership algorithm it is sufficient to assume that a message contains the broadcaster's local view mem^t_b on the membership.

As the order of messages is statically defined there is no need for special membership messages. Instead, a successfully received message is interpreted as a life-sign of the sender and a receiver will maintain the broadcaster in its local membership set if it agrees with the broadcaster's critical state information and hence with its membership set. Conversely, if a processor does not receive an expected message or does not agree with the broadcaster's view on the membership, the broadcaster will be considered faulty and the receiver removes it from its membership set.

The group membership algorithm is designed to operate in the presence of faults. A processor can be *send-faulty*, in which case it will fail to broadcast in its next slot, while a receive-faulty processor will not succeed in receiving the message of the next non-faulty processor. We use NF^t to denote the set of non-faulty processors at time t and $p \notin NF^t$ indicates that p is either send-faulty or receive-faulty at time t. Furthermore, $sends^t_b$ describes that the current broadcaster b sends a message on the bus, while $arrives^t_p$ means that the message sent by the broadcaster arrives at the receiver p.

A non-faulty broadcaster b will only send a message if b is contained in its own membership set; if b has removed itself from the membership set (due to diagnosing a fault) it will stop sending message in its broadcast slot. The following specification[10] shows the axiomatization of $sends^t_b$ as defined in PVS:

NF^t : set[proc]
$ends^t_b$: bool
$Arrives^t_p$: bool
Sending : Axiom
LET b = broadcaster(t) IN
$b \in NF^t \wedge b \in mem^t_b \Rightarrow sends^t_b$
fail_silence : Axiom
LET b = broadcaster(t) IN
$b \notin mem^t_b \Rightarrow sends^t_b$

A message sent by the current broadcaster b will arrive at a non-faulty processor p . Of course, there is no generation of spontaneous messages and hence messages arrive only if they have been sent. These axioms also imply that broadcasts are consistent: a message arrives either at all non-faulty processors or, if the broadcaster is send-faulty, at none of them. The PVS specification is given as follows:

arrival : Axiom
LET b = broadcaster(t) IN
$Sends^t_b \wedge p \in NF^t \Rightarrow arrives^t_p$
Nonarrival : Axiom
LET b = broadcaster(*t*) IN
$\neg sends^t_b \Rightarrow \neg arrives^t_p$

The task of a group membership algorithm is to diagnose the failure of a faulty processor and to inform all non-faulty processors about it. In order to cause a broadcaster to realize that it is send-faulty the TTP group membership algorithm uses an (implicit) acknowledgment mechanism. A processor p that is the broadcaster in slot t checks whether the next non-faulty broadcaster, say q, that sends in the next slot has the same membership set as q and in particular contains p in its membership set. If so, p can conclude that its broadcast was successful. Otherwise, either p failed to broadcast or q is receive-faulty. To resolve this ambiguity p waits for the next non-faulty broadcaster following q, say r. If r contains p in its membership set but not q while having the same view considering other processors, the original message of p

was sent correctly and q failed. If p is not in r's membership set, but q is (and the rest of the membership sets of p and r are the same), then q and r agree that p failed to send. In this case, p will remove itself from its own membership set and fail silently.

A similar mechanism could be used for diagnosing receive faults: If a processor p does not receive an expected message it could check whether the next non-faulty broadcaster maintained the original sender in its membership set in which case p must realize that it has suffered from a receive fault. However, TTP employs a slightly different mechanism that is also used to avoid the formation of disjoint cliques at the same time. A clique is a group of processors where agreement on the current state is reached only within the group. Each processor p maintains two counters, acc^t_p and rej^t_p, which keep track of how many messages p has accepted (successfully received) and rejected, respectively. A processor p will increment the counter rej^t_p if p does not agree with the broadcaster's view on the membership. In p's next broadcast slot it checks whether it has accepted more messages in the last round than it has rejected. If so, p resets the counters and broadcasts; the other case indicates that p suffered from a receive fault and therefore p removes itself from the membership and by not broadcasting its message, p can inform the other processors about its failure.

Formally, the group membership algorithm is described by a set of guarded commands. In every slot t, every processor executes exactly one of these commands. The guards are evaluated in a top-down order. The formal description involves two additional boolean state variables, $prev^t_p$ and $doubt^t_p$. If a processor p was the previous broadcaster and now waits for being acknowledged, prevp is set to true, while doubtp is true if p did not get acknowledged by its successor and waits for the second successor to resolve the conflict. In this case, the variable $succ^t_p$ holds p's first successor which refused to acknowledge p. In the following definition state components that are not mentioned explicitly do not change.

Reintegration: In the algorithm described previously in 4.1, a node (processor) reintegration, after it has been detected faulty (in receive or transmission fault), has not been studied. We consider that a processor Pi is faulty if Pi doesn't appear anymore in views of other non faulty processors. This transient fault affects processor Pi instantly. It means that the faulty processor Pi may reintegrate the group in the next slot (i.e. slot (i+1) mod N). Because the transient fault has disappeared, Pi is now non faulty and is said *reintegrator* processor. Any processor is either broadcaster one or receiver one.

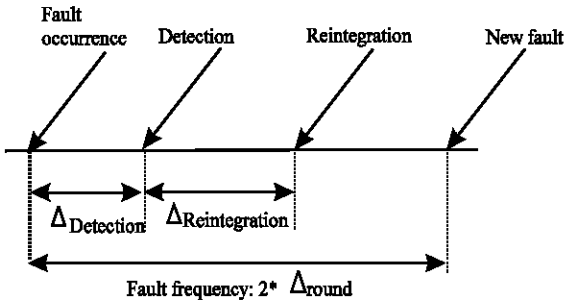

Fig. 3: Hypothesis of fault frequency

- Broadcaster: the processor Pi can broadcast a message at slot i modulo N.
- Receiver: Pi can only receive a message at slot j (j # i).

In order to study the reintegration protocol, byzantine faults are assumed to be masked by TTA[2]. Furthermore, these faults are not included in the basic fault hypothesis (section 4). The protocol is exclusively modeled for symetric faults and in TTA specification, only a single fault may occur in two TDMA rounds (Fig. 3).

$$\Delta_{\text{Détection}} + \Delta_{\text{Réintégration}} \leq 2^* \Delta_{\text{round}}$$

GMP algorithm with a node reintegration: Formally, the group membership algorithm is described by a set of guarded commands. In every slot, every processor executes exactly one of these commands. The guards are evaluated in a top-down order and the first command whose guard is evaluated to true be executed. The formal description involves one additional boolean state variable, $integrat^t_p$. If a processor p was detected faulty $integrat_p$ is set to true. The following formal definitions list 20 such guarded commands, with two of them, namely the clauses (1) and (2), describing the behaviour of the current broadcaster and the remaining eighteen commands consider the receivers. Among the latter, we can identify four sub-categories: clause (3) deals with a supposedly faulty processor that has already removed itself from its membership set, the clauses (4) to (10) describe the behaviour of a processor that has broadcast a message and waits for acknowledgment. The clauses (11) to (14) deal with a processor that has not been acknowledgment by its first successor and waits for the second successor to disambiguate the situation and finally, the clauses (15) to (20) comprise all other receiving processors.

In a normal situation the processor that is the broadcaster in the current slot executes the clause (1). In the exception to this ordinary behaviour of the broadcaster, the processor does not agree with the majority of the processors then it will have rejected more messages of the previous TDMA round and hence the rej counter will be greater than the acc counter. In this case the broadcaster must not send and removes itself from its membership set (clause (2)).

The clause (3) describes the behaviour of a processor that has already removed itself from its membership set. Such a processor will be reintegrated to the group but not immediately. It resets the *integrat* flag to true and the counters acc and rej to the values of 2 and 0, respectively and its membership set will contain only itself and the current broadcaster.

The clauses (4) and (6) describe the behaviour of the processor that has just been the broadcaster in the previous slot, that is, has the *prev* flag set. The first case describes the situation when the processor, that has the integrat flag set, receives a correct message (the boolean expression $arrive_p^t$) and the current broadcaster has accepted the previous broadcaster's message. Therefore, the processor can finish the acknowledgment process and reset the prev and integrat flags to false. Moreover, because the last message was accepted, the acc_p^t counter is increased.

The clause (5) describes the behaviour of an integrator processor, that without inspecting its membership set (it is a faulty previous broadcaster), receives a correct message. Therefore, the processor can finish the acknowledgment process, inserts the current broadcaster in its membership, reset the prev flag to false and increases corresponding counter.

If the previous broadcaster has received a negative acknowledgment from its successor it has to examine another processor's membership on the correctness of the original message transmission in order to resolve the conflict whether it committed a send fault or its first successor suffered from a receive fault. Such processor will have the doubt flag set to true (clauses (11) to (14)). The clauses (15) and (18) describe the behaviour when the processor receives a message and agrees with the broadcaster's view on the membership. The receiver is either a reintegrator processor or ordinary receiver.

The clause (17) is evaluated to true when the processor receivers a message and agrees with the integrator broadcaster's view on the membership.

The final three commands in the description of the state transition function for the group membership deal with the ordinary receivers, which do not analyze the received message for acknowledgment purposes.

broadcaster

(1) $acc_p^t > rej_p^t \wedge acc_p^t \geq 2 \rightarrow mem_p^{t+1} = mem_p^t \wedge prev_p^{t+1} = T \wedge acc_p^{t+1} = 1 \wedge rej_p^{t+1} = 0$

(2) otherwise $\rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\}$

Receiver

(3) $p \notin mem_p^t \rightarrow integrat_p^{t+1} = T \wedge mem_p^{t+1} = \{p,b\} \wedge acc_p^{t+1} = 2 \wedge rej_p^{t+1} = 0$

(4) $prev_p^t \wedge arrive_p^t \wedge mem_b^t = mem_p^t \cup \{p\} \wedge integrat_p^t \rightarrow mem_p^{t+1} = mem_p^t \wedge prev_p^{t+1} = F \wedge acc_p^{t+1} = acc_p^t + 1 \wedge integrat_p^{t+1} = F$

(5) $prev_p^t \wedge arrive_p^t \wedge integrat_p^t \rightarrow mem_p^{t+1} = mem_p^t \cup \{b\} \wedge prev_p^{t+1} = F \wedge acc_p^{t+1} = acc_p^t + 1$

(6) $prev_p^t \wedge arrive_p^t \wedge mem_b^t = mem_p^t \cup \{p\} \rightarrow mem_p^{t+1} = mem_p^t \wedge prev_p^{t+1} = F \wedge acc_p^{t+1} = acc_p^t + 1$

(7) $prev_p^t \wedge arrive_p^t \wedge mem_b^t = mem_p^t \setminus \{p\} \rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\} \wedge prev_p^{t+1} = F \wedge doubt_p^{t+1} = T \wedge rej_p^{t+1} = rej_p^t + 1 \wedge succ_p^{t+1} = b$

(8) $prev_p^t \wedge arrive_p^t \wedge integrat_b^t \rightarrow mem_p^{t+1} = mem_p^t \cup \{b\} \wedge acc_p^{t+1} = acc_p^t + 1 \wedge prev_p^{t+1} = F$

(9) $prev_p^t \wedge null_p^t \rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\}$

(10) $prev_p^t \rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\} \wedge rej_p^{t+1} = rej_p^t + 1$

(11) $doubt_p^t \wedge arrive_p^t \wedge mem_b^t = mem_p^t \cup \{p\} \setminus \{succ_p^t\} \rightarrow mem_p^{t+1} = mem_p^t \wedge acc_p^{t+1} = acc_p^t + 1 \wedge doubt_p^{t+1} = F$

(12) $doubt_p^t \wedge arrive_p^t \wedge mem_b^t = mem_p^t \cup \{succ_p^t, b\} \setminus \{p\} \rightarrow mem_p^{t+1} = mem_p^t \cup \{succ_p^t\} \setminus \{p\} \wedge doubt_p^{t+1} = F \wedge acc_p^{t+1} = acc_p^t + 1$

(13) $doubt_p^t \wedge null_p^t \rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\}$

(14) $doubt_p^t \rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\} \wedge rej_p^{t+1} = rej_p^t + 1$

(15) $arrive_p^t \wedge integrat_p^t$

$$\wedge\ (mem_p^t = mem_b^t) \rightarrow mem_p^{t+1} = mem_p^t$$
$$\wedge\ acc_p^{t+1} = acc_p^t + 1$$
$$\wedge\ integrat_p^{t+1} = F$$

(16) $arrive_p^t \wedge integrat_p^t \rightarrow mem_p^{t+1} = mem_p^t \cup \{b\}$
$$\wedge\ acc_p^{t+1} = acc_p^t + 1$$

(17) $arrive_p^t \wedge integrat_b^t \rightarrow mem_p^{t+1} = mem_p^t \cup \{b\}$
$$\wedge\ acc_p^{t+1} = acc_p^t + 1$$

(18) $arrive_p^t$
$$\wedge\ (mem_p^t = mem_b^t) \rightarrow mem_p^{t+1} = mem_p^t$$
$$\wedge\ acc_p^{t+1} = acc_p^t + 1$$

(19) $null_p^t \rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\}$

(20) otherwise $\rightarrow mem_p^{t+1} = mem_p^t \setminus \{b\}$
$$\wedge\ rej_p^{t+1} = rej_p^t + 1$$

CONCLUSION

In the GMP protocol of TTP/C of the TTA architecture, any detected faulty node, is immediately excluded from the group. This gradual exclusion process risks invalidating the protocol after N-3 successive failures if the ability of faulty node reintegration is not implemented. Our contribution in this paper is to remedy this serious problem. Therefore, we have proposed a formal framework to model the group membership protocol with nodes reintegration. This additional aspect allows GMP protocol to get more reliability in the context of critical embedded applications. Our future work will concern the verification of the model with the well known PVS theorem prover.

REFENECES

1. Kopetz, H., 1995. The time-triggered approach to real-time system design. In B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood, editors, Predictably Dependable Computing Systems, Basic Research Series. Springer-Verlag.
2. Kopetz, H. and G. Bauer, 2002. The time-triggered architecture. Special Issue of IEEE on Modeling and Design of Embedded Software.
3. Heiner, G. and T. Thurner, 1998. Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems. Proc. 28th Intl. Symp. on Fault-Tolerant Computing. IEEE Computer Society.
4. Owre, S., J. Rushby, N. Shankar and F. von Henke, 1995. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. IEEE Trans. on Software Engineering, 21: 107-125.
5. Specification of the TTP/C Protocol, 2000. Available on request from TTTech,. http://www.tttech.com .
6. Time-Triggered, 2002. Protocol TTP/C High-Level Specification Document. Available on request from TTTech at http://www.tttech.com/technology/specrequest.html.
7. Kopetz, H. and G. Grünsteidl, 1994. TTP-a time-triggered protocol for fault-tolerant real-time systems. IEEE Computer, 27: 14-23.
8. Bauer, G., H. Kopetz and W. Steiner, 2002. Byzantine Fault Containment in TTP/C. Proc. Intl. Workshop on Real-Time LANs in the Internet age, pp: 13-16.
9. Pfeifer, H., D. Schwier and F.W. von Henke, 1999. Formal Verification for Time-Triggered Clock Synchronization. In Charles B. Weinstock and John Rushby (Eds.), Editors, 7th Dependable Computing for Critical Applications (DCCA'99), volume 12 of Dependable Computing and Fault-Tolerant Systems, IEEE Computer Society, pp: 207-226,
10. Pfeifer, H., 2000. Formal verification of the TTP group membership algorithm. In Tommaso Bolognesi and DiegoLatella, editors, Formal Methods for Distributed System Development Proceedings of FORTE XIII/ PSTV XX 2000, Pisa, Italy, Kluwer Academic Publishers, pp: 3-18.
11. Steiner, W., J. Rushby, M. Sorea and H. Pfeifer, 2004. Model Checking a Fault-Tolerant Startup Algrithm: From Design Exploration To Exhaustive fault Simulation. In Proc. Conf. on Dependable Systems and Networks. IEEE Computer Society.
12. Rushby, J., 2002. An Overview of Formal Verification for Time-Triggered Architecture. Proc. 7th Intl. Symp. On Formal Techniques in Real-Time and Fault-Tolerant Systems. Volume 2469 of LNCS, Springer-Verlag, pp: 83-105.
13. Kopetz, H., 1998. The time-triggered Model of Computation. Real Time-System Symposium, Madrid, Spain, IEEE Computer Society Press.
14. Pfeifer, H., 2003. Formal Analysis of Fault-Tolerant Algorithms in the Time-Triggered Architecture. PhD thesis, Universität Ulm, Germany.