Journal of Computer Science 3 (1): 28-34, 2007
ISSN 1549-3636

Formal Analysis of Fault-tolerant Algorithm in the Time-triggered Architecture

[1]Aliouat Zibouda , [1]Aliouat Makhlouf and [2]Batouche Chawki
[1]Computer Science Department, University of Ferhat Abbes, Faculty of Engineer Science
Sétif 19000 Algeria,
[2] Computer Science Department, Faculty of Engineer Science, University of Constantine
Contantine 25000 Algeria

Abstract: Time-Triggered architecture (TTA) provides a computing infrastructure for the design and implementation of dependable distributed systems. The core building block of the TTA is the communication protocol TTP/C. This protocol has been designed to provide no faulty nodes. TTP/C integrates a set of fault-tolerant services like: message transmissions, clocks synchronization and Group Membership Protocol (GMP). The GMP protocol ensures that each TTA node maintains a private membership set, which records all the nodes that are believed to be nonfaulty. In the GMP protocol previously studied in the literature, any detected faulty node was 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 was not implemented. Our contribution in this paper was to remedy this serious problem. A node reintegration increases system survivability by allowing a (recovering) transiently-faulty node to regain a group. Our proposal algorithm, devoted to node reintegration inside the group membership protocol, was formally specified and verified using a diagrammatic representation. The verification of the proposal has been checked with the well known PVS theorem prover.

Keywords: Time-triggered Architecture, TTP/C, GMP, deductive verification, reintegration

INTRODUCTION

The Time-Triggered Architecture (TTA) is distributed computer architecture for the implementation of highly dependable real-time systems. TTA is intended for devices controlling safety-critical electronic systems without mechanical backup, so-called “by-wire” systems such as those for automotive steering, braking and suspension control[1]. It has been argued that the kind of reliability required in such situations cannot be achieved without a careful formal analysis of the mechanisms and algorithms involved[2,3]. A TTA system is provided with fault tolerance abilities implemented in both hardware and software components. Whereas the hardware relies on redundant nodes and duplicated communication channels, the software uses algorithms that control such basic services as membership agreement, clique avoidance and clock synchronization that are accomplished by TTP/C. The Time-Triggered Protocol TTP/C is the core of the communication level of TTA. Furthermore, TTP/C is designed to provide an acceptable level of fault tolerance. In particular, the protocol has to ensure that non-faulty nodes receive consistent data despite the presence of possibly faulty nodes or a faulty communication channel. The provision of fault tolerance is based on a number of assumptions, which constitute the so-called *fault hypothesis*. The main assumption for the algorithms implemented in TTP/C is that a fault manifests itself as either a reception fault or a consistent send fault of some node[4]. Especially, TTP/C assumes that transmission faults are consistent, that is, messages are received by either all non-faulty nodes or none.

In this regard, this paper is devoted to the transient failures. The faulty node that has been excluded can reintegrate the group membership after recovery operation. For this, a detailed overview of formal analysis work for the reintegration node is given in[5]. we concentrate our attention, in this paper, to the formal verification of a node reintegration in the TTP/C group membership protocol, thereby complementing and extending previous work[5,6]. The group membership algorithm[6] is modeled as a synchronous system. Its verification is significantly more difficult than other fault-tolerant algorithms because information about the failure of processors is not available immediately but only with a certain delay. Therefore one has to be very careful when reasoning about possibly failed components. Verification of safety properties, like the requirement that all (non-faulty) processors of a system should have the same view about the current

Corresponding Author: Aliouat Zibouda University of Ferhat Abbes, Faculty of Engineer Science, Computer Science Department, Setif 19000 Algeria.

membership status of other processors, is typically accomplished by an induction proof.

In order to establish the induction step, however, one generally has to strengthen the invariant because often enough the property of interest is not inductive. Usually, repeated strengthening is necessary before an inductive invariant is found and although some of the strengthening can be generated automatically, this becomes the main task when performing a mechanized verification. We take an approach proposed by Rushby[7] instead of expressing the correctness property as one large conjunction, we use a set of disjunctively connected formulas that can be seen as the description of an abstract state machine[7]. Each disjunction contains the desired property and represents a particular configuration the membership algorithm can reach.

To establish the correctness of the algorithm, one has to show that at every point in time, the system is in one of these configurations. For the TTP group membership algorithm, we have formally proved both an agreement property, i.e., that *every non-faulty node considers the same set of processors to be part of the membership* and a self-diagnosis-reintegration property that states: *a faulty processor will eventually remove itself from the membership and it will be re-integrated to the group no later than 3n + 1 steps after the fault occurred* (n is the total number of processors involved in GMP protocol). All definitions and proofs have been developed and mechanically checked with the PVS theorem prover system[6].

GROUP MEMBERSHIP PROTOCOL

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. 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 this type of 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. We assume that the fault occurrences 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[8].

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.

$\forall p \in NF^t : mem^t_p = NF^t \vee \exists x \notin NF^t : mem^t_p = NF^t \cup \{x\}$

$\vee p \notin NF^t : mem^t_p = \varnothing\ mem^t_p \subseteq NF^t \vee \{p\}$

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

$\forall p,q \in NF^t : mem^t_p = mem^t_q$

* **Self-diagnosis-reintegration in limited Time:** A processor that becomes faulty should eventually diagnose its fault; empty its membership set and reintegrate to the group in no more than 3n slots + 1.

$\forall x: x \in NF^t \wedge x \notin NF^{t+1} \Rightarrow \exists s: 0<s \wedge s \leq 3n+1 \wedge x \in NF^{t+s}$

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 data message 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 to be sent. 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.

At all slots, there is a broadcaster b will have to send a message. Of course, a broadcaster can be faulty (integrator) or non-faulty. The integrator has already added itself in its own membership set. The following specification shows the axiomatization of $sends^t_b$ as we defined it in PVS:

$\mathcal{NF}^t$: `set[proc]`
$Sends^t_b$: `bool`
$Arrives^t_p$: `bool`
Integrat^t_p : `bool`
Sending : Axiom
`LET` b = `broadcaster(`t`) IN`
$b \in \mathrm{mem}^t_b \Rightarrow sends^t_b$

A message sent by the current broadcaster b will arrive at a non-faulty processor p and also to the integrator processor. 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`
$sends^t_b \wedge p \in \mathcal{NF}^t \Rightarrow arrives^t_p$
arrival_int : Axiom
`LET b = broadcaster(t) IN`
$sends^t_b \wedge \mathrm{integrat}^t_p \Rightarrow arrives^t_p$
nonarrival : Axiom
`LET b = broadcaster(t) IN`
$\neg sends^t_b \Rightarrow \neg arrives^t_p$

The processor that has been detected failed and has emptied its membership set will be integrator. The following specification shows the axiomatization of *Integrating* as defined in PVS:

Integrating : Axiom
`LET b = broadcaster(t) IN`
$\mathrm{mem}^t_p = empty \neg integrat^t_p \wedge \Rightarrow integrat^{t+1}_p$

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 will send 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 is 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 then sent correctly and q is 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 is 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* c is a group g (g is a subset of global set of processors) of processors where agreement on the current state is reached only within the group g.

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, p checks whether it has accepted more messages in the last round than it has rejected. If so, p resets its counters and broadcasts; the other case indicates that p suffered from a receive fault. Therefore, p removes itself from the membership set 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, prev_p is set to true, while doubt_p 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.

FORMAL DESCRIPTION OF THE GMP ALGORITHM

The formal description of the GMP algorithm needs one additional boolean state variable, $integrat^t_p$. If a processor p was detected faulty $integrat^t_p$ 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 acknowledged by its first successor and waits for the second successor to disambiguate the situation. 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 (This broadcaster does not agree with other processors that is, it has rejected more messages in the previous TDMA round and hence its *rej* counter is greater than its *acc* counter), the broadcaster must not send and empties its membership set (clause (2)).

The clause (3) describes the behaviour of a processor that has already emptied 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. Its membership set will then 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 is set (true). The first case (clause (4)) describes the situation when the processor, that has the *integrat* flag set, receives a correct message (the boolean expression $arrive_p^t$ is true) 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, resets the *prev* flag to false and increases corresponding counter.

If the previous broadcaster has received a negative acknowledgment from its successor (clause (7)), 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.

The clause (8) describes the behavior when the current broadcaster is integrator and the receiver is a previous broadcaster and has correctly received a message.

The clauses (11) to (14) concern a processor that has *doubt* flag set to true.

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 one.

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

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 $\rightarrow mem_p^{t+1} = emptyset \wedge prev_p^{t+1} = F \wedge acc_p^{t+1} = 0 \wedge rej_p^{t+1} = 0$

Receiver

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

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

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

APPROACH TO VERIFY THE PROPOSAL ALGORITHM

For the verification of the node reintegration part of TTP/C group membership algorithm we apply a method proposed by Rushby[7]. The requirements of *validity* and *agreement* express properties that should hold for all reachable states of the system. Such invariants, or safety properties, are usually verified by some form of induction proof. The configurations are defined such that every single configuration implies the desired property and to verify that property one has to show that at all times the system is in one of these configurations. Thus, the main part of the proof can be represented as a *configurations diagram*. The diagram for the group membership algorithm is shown in Fig. 1. The nodes of the diagram represent the *configurations* and arrows denote transitions from one configuration to others and are labeled with transition conditions. Configurations are parameterized by the time *t* and describe the global state the system is in. Configurations can have additional parameters such as processors (*x*, *y*, . . .) that behave differently from the rest of system, or additional entities necessary to describe the system state. The labels of transitions express the preconditions for the system to move from one configuration to another. For example, the label $x = z$ of the transition from *integration-ok* to *stable* means that the system takes this transition if *x* is the last broadcaster. The transition conditions leading from one configuration need not necessarily be disjoint, but one has to show that they are complete in the sense that their disjunction is true.

The diagram can be developed step-by-step. One usually starts by defining some initial configuration or the one in which the system stays under normal circumstances, i. e. as long as no fault occurs. For TTP, this central configuration is the one labeled *stable*. By symbolically evaluating the algorithm in the current configuration and by splitting on possible cases, we generate some new configurations and the transitions from the original configuration are labeled with the appropriate conditions. By repeatedly applying this construction on each transition and each new configuration, one aims to develop a closed diagram. To prove safety properties like *validity* or *agreement,* one has then to demonstrate that every configuration implies the desired property and that the disjunction of the transition conditions leading from any one configuration evaluates to true; this ensures that there is no other configuration the system can possibly get into. In order to prove liveness properties like *self-diagnosis-integration* one has to establish that the system can not loop forever on a configuration other than *stable*.

There are several benefits to this approach: firstly, the diagram can be developed incrementally and in a totally systematic way by symbolically executing one step of the algorithm in every configuration. Secondly, the completed diagram is a suitable mean of analyzing the difficult special cases of the algorithm and to explain how and why it works (or doesn't). Lastly, it seems that the creative steps in developing the proof can be accomplished easier than by using the traditional way of repeated invariant strengthening. The configurations as presented here generally are not invariants and are therefore identified more easily.

The next section describes how the configuration diagram for the TTP group membership algorithm is gradually developed and outlines the verification of the two correctness requirements for the TTP group membership algorithm.

DEVELOPING THE CONFIGURATION DIAGRAM

The system is said to be in a *stable* configuration if the membership set of all non-faulty processors *p* is equal to NF^t. The set of all non-faulty processors at time *t* and a faulty processor has already diagnosed its fault and thus removed itself from its own membership set. For *stable,* the two safety properties *validity* and *agreement* follow immediately from these definitions. Moreover, *stable* is the initial configuration of the system.

Stable (t, z) : bool =
recent (t, z)
$\wedge\ \forall p : p \in NF^t \Rightarrow mem_p^t = NF^t$
$\wedge\ p = z \Leftrightarrow acc_p^t > rej_p^t$
$\wedge\ p \neq z \Rightarrow acc_p^t > rej_p^t + 1$

$\wedge\ prev^t_p = T \Leftrightarrow p = z$
$\wedge\ doubt^t_p = F$
$\wedge\ integrat^t_p = F$

In the configuration *stable*(*t, z*) the counters of non-faulty processors are set such that $acc^t_p > rej^t_p+1$.

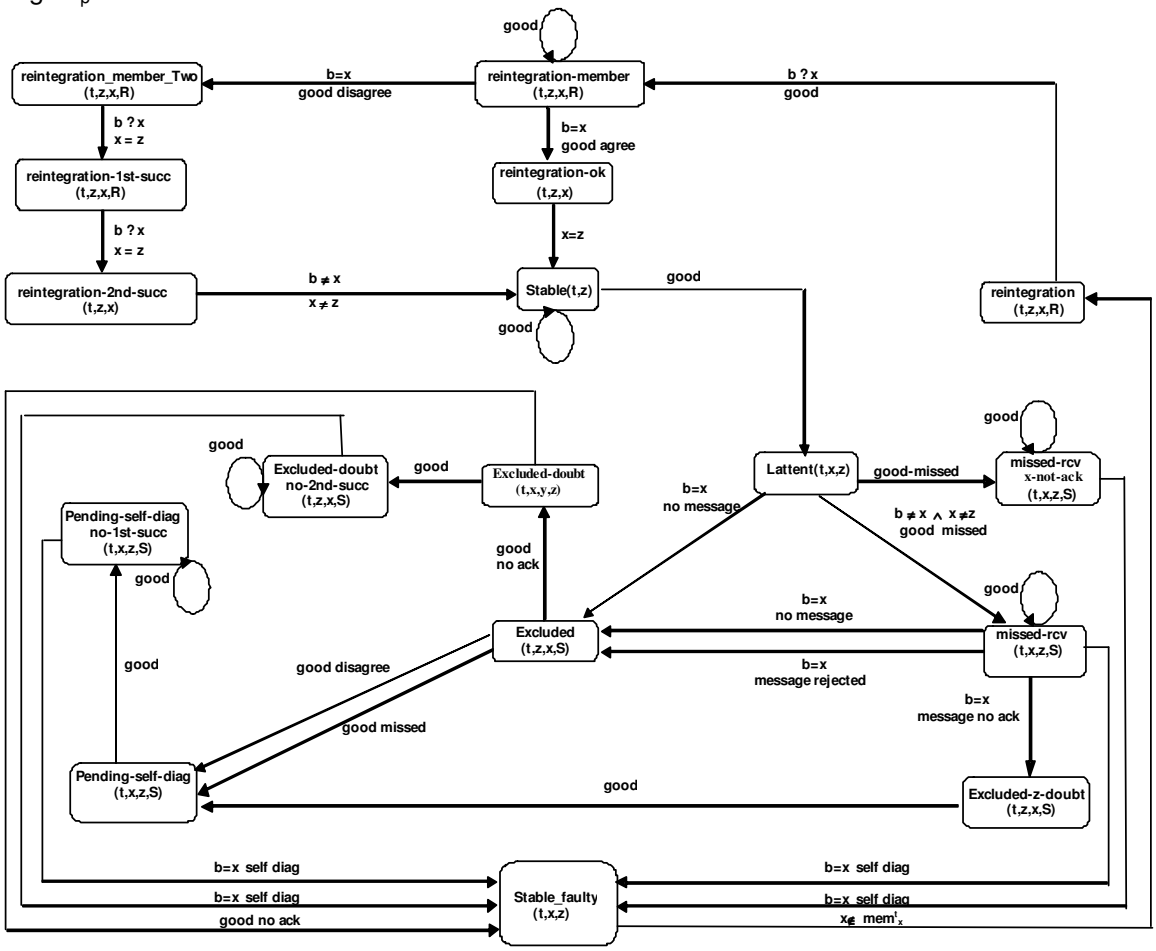

Fig. 1: Configuration diagram for the Global TTP membership algorithm

This is to allow for a non-faulty processor *p* to cope with a send fault of other broadcaster in the next round, in which case, the counter $rej^t p$ will be increased; this should not lead to *p* removing itself from its own membership set in its next sending slot, for which $acc^t_p > rej^t_p$ must hold. However, the most recent non-faulty broadcaster, say *z*, cannot satisfy this condition as in its sending slot *z,* sets the counters: $acc^t_z = 1$ and $rej^t_z = 0$.

The expression *recent (t, z)* denotes that at time t, processor z is the recent non-faulty broadcaster.

The group membership algorithm is said to be in the *stable-faulty* configuration at time t, for t > 0, if the membership set of all non faulty processors is equal to NF^t and the faulty processor x has its membership set empty.

stable-faulty (t, x, z) : bool =
$x \notin NF^t \wedge$ recent (t, z)
$\wedge\ \forall$ p : before (t, z, p) $\Rightarrow p \notin NF^t$
$\wedge\ \forall$ p : (p $\in NF^t \vee$ p = x) $\Rightarrow mem^t_p = \varnothing \Leftrightarrow p = x$
$\wedge\ mem^t_p = NF^t \Leftrightarrow p \neq x$
$\wedge\ acc^t_p = rej^t_p + 1 \Leftrightarrow (p = z \vee p = x)$
$\wedge\ integrat^t_p = F$
$\wedge\ prev^t_p = T \Leftrightarrow p = z$
$\wedge\ doubt^t_p = F$

If a processor, say x, which was detected faulty and has its membership set empty becomes integrator, the system will be into a new configuration, that we call *reintegration.*

reintegration (t, x, z, R) : bool =
$x \notin NF^t \wedge z \in NF^t$
$\wedge\ \exists$ p : p $\in NF^t \wedge$ before (t, p, z)
$\wedge\ \forall$ p : (p $\in NF^t \vee$ p = x) $\Rightarrow mem^t_p = R \Leftrightarrow p = x$
$\wedge\ acc^t_p > rej^t_p + 1 \Leftrightarrow p = x$
$\wedge\ acc^t_p = rej^t_p + 1 \Leftrightarrow p = z$
$\wedge\ integrat^t_p = T \Leftrightarrow p = x$
$\wedge\ prev^t_p = T \Leftrightarrow p = z$
$\wedge\ doubt^t_p = F$

If the group membership algorithm is in stable-faulty configuration at time t with respect to x and z, then the agreement property holds at time t. The system

transits into the stable-faulty configuration if x is the current broadcaster but fails to send a message through command (2). Thus, the non-faulty processors remove x from their membership sets by executing the command (19) or (9) in the case of z. Therefore, all non-faulty processors have the same membership sets. The correctness property self-diagnosis-reintegration is a liveness property that is, once has left *stable* configuration, it can not be trapped in one of other configurations. It must be return to *stable* state no later than 3n+1 steps after the fault occurred.

Let the system be in the *stable-faulty* configuration at time t with respect to x and z and let b denoted the broadcaster at time t. If membership set's x is empty and if no new fault occurs in the next step then the system will be in the re*integration* configuration at time t + 1 with respect to x, b and the set {x, b}.

Stable-faulty to reintegration : LEMMA
LET b = broadcaster(t) IN
Stable-faulty(t,x,z) $\wedge$ b $\in$ $NF^t \wedge NF^t = NF^{t+1} \Rightarrow$ reintegration (t+1, x, b, {x, b})

CONCLUSION

In the previous 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 part allows GMP protocol to get more availability in the context of critical embedded applications.

The proofs of the main correctness properties of the algorithm have been developed and mechanically checked with the assistance of the PVS specification and verification system.

Further research is concerned with formally specifying the startup algorithm and finding ways to clearly identify the relationships and interfaces between group membership, startup and clock synchronization services.

REFERENCES

1. 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. Vol. 2469 of LNCS, Springer-Verlag, pp: 83-105.
2. Kopetz, H. and G. Bauer, 2002. The time-triggered architecture. Special Issue of IEEE on Modeling and Design of Embedded Software.
3. Anonymous, 2002. Time-Triggered Protocol TTP/C High-Level Specification Document. Available on request from TTTech at http://www.tttech.com/technology/ specrequest. html.
4. 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.
5. Aliouat, Z., 2006. Formal Modeling and analysis of a node Reintegration in the Time-Triggered Architecture. Asian J. Inform. Technol., 5: 706-711.
6. Pfeifer, H., 2000. Formal Verification of the TTP Group Membership Algorithm. In Tommaso Bolognesi and DiegoLatella, (Eds.), Formal Methods for Distributed System Development Proceedings of FORTE XIII / PSTV XX 2000, Pisa, Italy, Kluwer Academic Publishers, pp: 3-18.
7. Rushby, J., 2000. Verification Diagrams Revisited: Disjunctive Invariants for Easy Verification. In E. A. Emerson and A. P. Sistla, (Eds.), *Computer Aided Verification (CAV 2000)*, volume 1855 of *LNCS*, pages 508–520, Chicago, IL, Springer-Verlag.
8. Pfeifer, H., 2003. Formal analysis of fault-tolerant algorithms in the time-triggered architecture. Ph.D. Thesis, Universität Ulm, Germany.