Flexible, Cost-Effective Membership Agreement in Synchronous Systems

Raul Barbosa and Johan Karlsson
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{rbarbosa, johan}@ce.chalmers.se

Abstract

This paper presents a processor group membership protocol for fault-tolerant distributed real-time systems that utilize periodic, time-triggered scheduling for sending messages over the system's communication network. The protocol allows fault-free nodes to reach agreement on the operational state of all nodes in the presence of fail-silent *or* fail-reporting *node failures as well as network failures (lost or corrupted messages). The protocol is based on the principle that each message sent by a node in the membership is acknowledged by k other nodes in a system of n nodes, where k can be set to any number between 2 and $n-1$. Agreement on node failure (membership departure) and agreement on node recovery (membership reintegration) are handled by two different mechanisms. Agreement on departure is guaranteed if no more than $f = k-1$ failures occur in the same communication round, while at most one node can be reintegrated into the membership per communication round.*

1 Introduction

Group membership is often a cornerstone of the design of fault-tolerant distributed systems. The entities, or nodes, which constitute such systems, must cooperate in providing a correct service to its user(s), even in the presence of faults [2]. A correct behavior of the system requires that the non-faulty nodes have access to timely and consistent information about the operational state of all nodes. The algorithms devised to maintain this information, preferably in a decentralized manner, are usually known as processor group membership protocols.

This paper proposes a processor group membership protocol appropriate for hard real-time systems which utilize buses such as TTCAN [11] or FlexRay [3]. Such systems impose hard deadlines for achieving consensus on membership changes, while they offer limited bandwidth for the implementation of the membership services.

The numerous solutions proposed in the past reflect the diversity of situations where membership agreement services are of use. In this paper we consider group membership for systems relying on synchronous communication, where messages are transmitted within a known amount of time and processing nodes have a global notion of time. The membership problem in such systems was first described in [6]. Specially tailored solutions, for different applications, were subsequently devised and proposed in the literature [15], [14], [5] and [20]. The problem has also been widely studied in the context of asynchronous systems, which do not observe the synchrony hypothesis (e.g. [19], [9], [8] and [18]). The timed asynchronous model [7] combines these two classes of systems by assuming that all services, including membership, are timed although interprocess communication time is unbounded.

The protocol described in this paper assumes that each node in the system periodically sends messages that reach all other nodes under fault-free circumstances. It piggybacks all protocol-specific information on such periodic messages. The protocol allows fault-free nodes to reach agreement on the operational state of all nodes in the presence of node failures as well as network failures (lost or corrupted messages).

It is based on the principle that each message sent by a node which is present in the membership is acknowledged by k other nodes in a system of n nodes, where k can be set to any number between 2 and $n-1$. Each node must acknowledge k messages, and thus each message must contain k acknowledgement bits. The number of acknowledgement bits determines the number of simultaneous failures that can be tolerated in one communication round. Therefore, our protocol contrasts to most solutions intended for hard real-time systems in its flexibility to configure the trade-off between reliability and communication overhead, at design time.

Moreover, agreement on node failure (membership departure) and agreement on node recovery (membership reintegration) are handled by two different mechanisms. This

12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)
0-7695-2724-8/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY

allows the cost of fault-tolerant reintegration to be decreased by implementing the reintegration procedure using the event-driven fraction of the bus schedule, where available. Time-triggered buses such as FlexRay and TTCAN provide event-driven communication slots in addition to the static schedule, for which nodes can compete in response to events.

Agreement on departure is guaranteed if no more than $f = k - 1$ failures occur in the same communication round, while at most one node can be reintegrated into the group of operational nodes (the membership) per communication round. In our scheme, listening on the bus is enough for a restarted node to recover the state of the membership, before attempting reintegration.

Furthermore, the protocol maintains consistent views of the membership even when multiple permanent node failures occur simultaneously, if at least two nodes remain fault-free and no other types of failure than permanent node failures occur in the same communication round.

In addition to the protocol for processor group membership agreement, this paper presents the outline of an extension which keeps track of application process failures. This feature is intended for distributed real-time systems where the processing nodes offer effective fault-containment between different application processes executed on the node. In such systems the membership service should preferably handle both node failures and application process failures.

2 System Model

We consider a set of processing nodes linked by a time-triggered broadcast network. We assume that the network has either as bus or star topology. Processors (nodes) have their clocks tightly synchronized and send messages at pre-defined time-slots according to a time-triggered cyclic schedule. Each node has a single dedicated slot on the network which it uses to broadcast its messages in every communication round. Processing nodes are assumed to be *fail-silent*, i.e. either correct results or no results are broadcasted, or *fail-reporting*, i.e. either the correct result or a failure report (specifying the causes of failure) is produced.

In fault-free conditions the node which has the right to access the network (i.e. the sending node) does so by sending a message. The physical link then ensures this message is delivered to all other nodes (i.e. the receiving nodes). If, at some point, a node does not receive a valid message, due to loss or corruption, a failure has occurred. It could be a *failure of the sending node*, a *failure of the receiving node*, a *network failure* or a combination of these.

In our system model, we assume that failures can occur in the nodes, their incoming and outgoing links (protocol processors which provide the interface to the network), and the network itself. To simplify the discussion about the kind of failures our protocol can handle, we map these failure types into four different failure classes according their persistence, *permanent* or *transient*, and whether their impact on the system is *consistent* or *inconsistent*. We assume that a *transient failure* only affects one message. If several consecutive messages are lost, for instance, due to electromagnetic interference on the network, then we consider this as a case of multiple transient failures. A *permanent failure* will remain in the system until it is repaired, and may affect one node, or its outgoing or incoming link, or a point-to-point connection between a node and the hub if the network has a star topology. (A permanent failure of a non-redundant bus network will lead to a failure of the entire system, and is thus not relevant for our group membership protocol.)

We classify a failure as consistent if all receiving nodes fail to receive a message correctly, since all the receiving nodes in this case will have a consistent view of the system state. On the other hand, an inconsistent failure occurs if one node fails to receive a message correctly, since that node then has a different view of the system compared to all other nodes. We assume that a situation where some nodes receive a message correctly, and two or more nodes receive the message incorrectly, can only occur in presence of multiple failures. We do not include permanent failures of a receiving node in our failure model, since a permanent failure of a node can only be seen by other nodes when the faulty node is supposed to act as a sender. Table 1 shows how the different types of component failure are mapped into the different failure classes.

<table>
<tr><th></th><th>Sending Node</th><th>Outgoing Link</th><th>Network (Consistent)</th><th>Network (Inconsistent)</th><th>Incoming Link</th><th>Receiving Node</th></tr>
<tr><td>Transient</td><td colspan="3">1. Transient Consistent Failure</td><td colspan="3">2. Transient Inconsistent Failure</td></tr>
<tr><td>Permanent</td><td colspan="3">3. Permanent Consistent Failure</td><td colspan="2">4. Permanent Inconsistent Failure</td><td></td></tr>
</table>

Table 1. Failure classes.

The rationale for this failure model is to have a clear definition of what we mean by a failure, as we express the fault tolerance capabilities of our protocol in terms of the number of simultaneous[1] failures the protocol can cope with. As previously explained, the number of simultaneous failures under which the protocol reaches agreement on the membership depends on the number of acknowledgement bits used. An example of another failure model used in conjunction with processor-group membership protocols for time-triggered systems can be found in [15].

From the viewpoint of healthy nodes, a failure of the sending node means missing at least one message from this node or receiving at least one failure report. In any case the failure will be consistently detected by all healthy nodes. The same can be assumed when an outgoing link failure of

[1] By simultaneous failures we mean failures that occur during one communication round.

IEEE COMPUTER SOCIETY

the sending node occurs. These two failure types can therefore be classified as consistent failures, i.e. class 1 or class 3, depending on their persistence.

On the other hand, when a receiving node suffers a transient failure it will miss a single message. A transient incoming link failure will also have the same consequence. These failures are classified together as transient inconsistent (class 2). When the incoming link of a single node becomes permanently faulty we classify it as a permanent inconsistent failure (class 4).

Modeling communication failures entails some considerations on the topology of the networks deployed in safety-critical systems [21]. We assume either the existence of *structural redundancy* (e.g. duplicate buses), as in TTP [16], or that the network is based on a *bus topology* or a *star topology*.

The network failure model is supported by the following analysis. We assume that applying structural redundancy will allow single transient failures to be masked by the physical layer. When the network uses a bus topology it is reasonable to assume that the probability of an error affecting only the propagation of the last few bits of a message, causing some nodes to receive the correct message while other nodes receive a corrupted version, is negligible. When the star topology is used, network failures in the connection between the sending node and the star hub will be detected by all receiving nodes. On the other hand, failures occurring in a receiving node's connection to the star hub will only be perceived by this node. We assume that the hub itself will not introduce changes to this failure model.

When all nodes miss a single message due to a transient network failure we have a failure of the transient consistent class. On the other hand, if only one receiving node misses a single message, a failure of the transient inconsistent class has occurred. When the network failure is permanent then either one node is not able to send messages (permanent consistent failure) or one node is not able to receive messages (permanent inconsistent failure).

3 Protocol Specification

Let N denote the totally ordered set of processing nodes $N_1, N_2, \ldots, N_n$. The totality condition is intuitively satisfied by the order in which nodes broadcast in every round, imposed by the network schedule. We exclude systems with $n \leq 2$ processing nodes, where membership agreement is trivially achieved by independent assessment, from our analysis.

The membership protocol relies on the periodic messages sent by each node to piggyback a sequence of acknowledgements. Each node will append k acknowledgement bits to its message, confirming (or refuting) the reception of each of the previous k messages *from the nodes in the membership*. A given node N_j will be removed from the *local membership view* of N_i if and only if N_i does not receive a message from N_j and no positive acknowledgement for this message is received from any of the *sponsors* of N_j.

A node is said to be *sponsoring* node N_j if, along with its message, the acknowledgement of the last message from N_j is attached. Only nodes in the membership are involved in sponsoring relations. Under normal conditions each node will have k sponsors (and will be sponsoring k nodes). Thus, k stipulates how resilient the protocol will be to near-coincident failures, at the expense of communication overhead.

The protocol requires $k = 2$ sponsors per node in its minimum configuration. Setting $k = 1$ does not allow the agreement to be maintained after a transient receiving failure of a sponsor, since the sponsor itself will erroneously remove its sponsored node from the local membership view.

Only one node can be reintegrated in each communication round. A node attempts reintegration simply by broadcasting a correct message. Given that failures can take place during this round there is the need for fault-tolerance in the reintegration process. Reintegration activities require therefore an additional bit to be appended to each message. We believe that the best solution is to use the event-driven fraction of the bus schedule – available in architectures such as TTCAN or FlexRay – to decrease the cost of the reliable response to reintegration events. Since we make no assumption on the existence of such a dynamic fraction of the schedule, a reintegration bit is added to each message in the static part of the schedule.

Upon the reception of a correct message in a slot expected to be left unoccupied (dedicated to a node out of the membership) all receiving nodes will set the reintegration bit of their subsequent message to $true$. When the evidence of a reintegration is received by a node but it has not received any message from a previously failed node then it must declare itself failed, since it will certainly disagree on the global membership view.

Reintegration activities will seldom take place. Hence it is reasonable to assume that the probability of a fault occurring whilst a reintegration attempt takes place is very low. However, if this probability is non-negligible, there is the need for fault-tolerant reintegration. It is also reasonable to assume that allowing a single reintegration in every communication round is sufficient. Section 3.6 presents some considerations on how to increase both the fault-tolerance and the functionality of the reintegration process at the expense of increasing the time needed for reintegration and the bandwidth overhead.

The following sections firstly endow with the essential definitions. The formal specification of the agreement on departure and reintegration is then presented, leading to a

IEEE COMPUTER SOCIETY

discussion on the configuration, reliability and functionality of the protocol.

3.1 Definitions

Each node has access to a global view of time. This is supplied by clock synchronization mechanisms. In what concerns the membership problem, the required granularity of time is the *slot*. Hence, each node uses the global variable $slot$. Each node contains also the local variable $membership$.

- The integer variable $slot$ represents the progression of time. This variable in no more than an integer counter of slots, starting at 1 and incrementing to infinity.
- The predicate $slot(i)$ is defined, for each synchronous step, as $true$ if and only if $i = slot \mod n$. It thus indicates which of the processing nodes N_1, N_2, ..., N_n, is the sending node in each bus slot.
- The variable $membership$ represents the local view of the membership set.

We assume the existence, at any time, of at least two *full-fledged nodes*, which agree on the membership and are not subject to failures which would break the agreement. Booting into such a configuration relies on the existence of a reliable start-up mechanism [4]. Periods of the execution where this assumption does not hold must be properly handled by blackout mechanisms, such as the ones existing in TTP. During temporary blackouts the nodes attempt to maintain the local system in a safe state while monitoring the membership. When other nodes start to recover it is possible to return to a normal operating mode.

Section 3.2 describes how the membership set is updated in response to every received/missed message from nodes in the membership. Section 3.3 then specifies how to handle messages received from nodes absent from the membership – reintegrations.

3.2 Agreement on Departure

Initially, the $membership$ contains all processing nodes. Each node has a boolean array $present$. The boolean element $present[i]$ will be set to $false$ when the message expected from N_i is not received. It will be set to $true$ when a positive acknowledgement referring to this message is received from one of the sponsors of N_i. Initially, all the elements of the $present$ array are set to $true$.

We define the predicate $sponsor(i, j)$ as $true$ if and only if N_i is sponsoring N_j. According to the definition of sponsor, N_i is sponsoring its k_s predecessors (in the order of broadcast) which are contained in the membership. Here we use k_s instead of k to define $sponsor(i, j)$. When, in a given slot s, the membership set contains n_s nodes and $n_s \leq k$, *a node is not expected to sponsor itself*. Hence, when $n_s \leq k$, each node will be sponsoring its $k_s = n_s - 1$ membership predecessors. Otherwise, $k_s = k$.

The predicate $lastSponsor(i, j)$ is defined to be $true$ if and only if node N_i is sponsoring N_j and the immediate successor of N_i in the $membership$ is not sponsoring N_j. This predicate essentially states if N_i is the last node to acknowledge the previous message from N_j.

The protocol progresses in discrete steps corresponding to the bus slots. At each step the sending node (N_i such that $slot(i) = true$) sends a message which contains an ack field. The ack field is an array containing k boolean variables or, simply put, it is the sequence of acknowledgement bits referring to the previous message from the k_s nodes sponsored by N_i.

The ack field is built by using the $present$ array. The sending node sets the ack bit which refers to a sponsored node N_a to $present[a]$. Thus it will be $true$ if and only if the sending node received the previous message from N_a or any of its acknowledgements.

When the slot of a node N_i in the $membership$ comes to an end, all nodes, including the sending node, synchronously execute the decision-making step. This is the time when the $membership$ set is updated in the following way: Node N_j, such that $lastSponsor(i, j) = true$, will be removed from the $membership$ if and only if $present[j] = false$.

In this decision-making step a node which verifies that no messages were received during the last round of communication (i.e. all elements of $present$ are $false$) has to remove itself from the $membership$ by declaring itself failed and sending no more messages. Since the existence of two nodes in the $membership$, at all times, is assumed, this node has certainly suffered a permanent fault which doesn't allow the reception of messages.

3.3 Agreement on Reintegration

The reintegration procedure commences when $slot(r) = true$, $N_r \notin membership$ and a message is correctly received by at least one node. This means that N_r is up and running again and the receiving nodes will set their local variable $reintegrating \leftarrow r$. This variable keeps track of the node which is currently reintegrating and is therefore initialized to $null$.

At each step the sending node (N_j such that $slot(j) = true$) piggybacks a $r\text{-}ack$ bit to its message. The $r\text{-}ack$ bit serves the purpose of notifying the nodes which might have lost the message from the reintegrating node. Therefore, a sending node will set $r\text{-}ack \leftarrow true$ if and only if $reintegrating \neq null$.

IEEE COMPUTER SOCIETY

Any node which receives a message reporting $r\text{-}ack = true$ when $reintegrating = null$ has to remove itself from the membership by declaring itself failed and sending no more messages. This node has certainly missed a reintegration message (from an unknown source) and will therefore be left out of the membership.

The synchronous decision-making step about the reintegration of node N_r is executed when $reintegrating = r$ (therefore $reintegrating \neq null$) and the slot of the first predecessor of N_r contained in the membership comes to an end. This is processed by i) adding N_r to the $membership$, ii) setting $reintegrating \leftarrow null$ and iii) $present[r] \leftarrow true$.

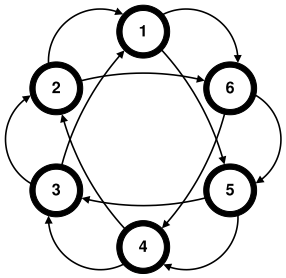

Figure 1. Sponsor-graph of a 6-node system ($k = 2$).

3.4 Recovery of Failed Nodes

When a previously failed node is able to restart, after a downtime period, it will need to recover the state of the membership before broadcasting any messages. We assume that fundamental data such as the bus schedule will survive the crash. The node only needs to recover the state of the membership, which is dynamic.

There is no need for explicit broadcast of the membership state since a restarting node may listen on the bus and infer which nodes are in the membership. If no failures occur, the messages received during two communication rounds and their appended ack and $r\text{-}ack$ flags permit the recovery of the membership state. The time for recovery is therefore unbounded since, for example, a transient failure of the restarting node may occur in every communication round.

Listening on the bus and analyzing the ack and $r\text{-}ack$ fields is also required to ensure that only one node is attempting reintegration at any given moment. If two nodes restart simultaneously, then it is possible for the first (in the order of communication) to reintegrate the membership, while the second waits for one communication round before sending any messages.

3.5 Message Overhead vs. Reliability

Choosing the number k of sponsors per node defines the balance between fault-tolerance and communication overhead. It is possible to identify the failure conditions which maintain the agreement by analyzing the directed graphs of sponsoring relations (sponsor-graphs). Figure 1 depicts such a sponsor-graph for an example of a 6-node system with $k = 2$.

A sponsor-graph is initialized with each vertex representing a node in the system. Initially, each directed edge from a given node N_i to N_j denotes that $sponsor(i, j) = true$. This graph evolves at every synchronous step (bus-slot) by removing an edge whenever the corresponding acknowledgement is missed by at least one node. When a decision-making slot comes and N_i is found to be faulty by all other nodes, then the sponsor-graph will be updated by removing the edges directed to vertex i and adding edges directed to the nodes which then obtained new sponsors (the previous sponsors of N_i are now sponsoring some other nodes).

Agreement will be maintained when all fault-free nodes receive at least one acknowledgement of any given message. Therefore the system is able to maintain the agreement if there is always at least one edge directed to each vertex.

Any failure which is not a permanent inconsistent failure will remove at most one edge directed to any given node. Therefore, if f such failures are to be tolerated in each communication round, the number of initial edges (i.e. the number of sponsors), should be $k = f + 1$. A permanent inconsistent failure will become apparent as a permanent consistent failure when the failed node declares itself failed and stops sending messages.

Agreement on the "real" state of the membership is reached k slots after the most recent failure. Consequently, at this point the system has redirected the sponsoring edges into a safe configuration again.

For any value of k the protocol maintains consistent views of the membership even when multiple permanent node failures occur, if this is the only type of failure to occur in the period between the first node failure and k slots after the last node failure. This type of failure is consistently perceived by all fault-free nodes. Therefore, all fault-free nodes will miss the messages from the permanently failed nodes and no positive acknowledgement on these messages will be sent. Agreement on the departure of the last node to suffer such a failure is reached k slots later.

The graph in Figure 1 exemplifies a sponsor-graph where no faults have occurred. When a failure of sending node 1 occurs (either transient or permanent) the graph will change into the one illustrated in Figure 2. Two slots later, after node 3 (the last sponsor of node 1) broadcasts, the system is "reconfigured" into the graph in Figure 3. In the critical

0-7695-2724-8/06 $20.00 © 2006 IEEE
IEEE COMPUTER SOCIETY

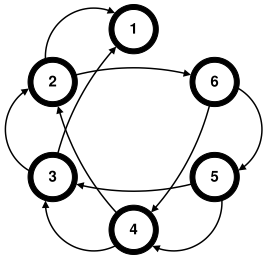

Figure 2. Sponsor-graph immediately after the failure of sending node 1.

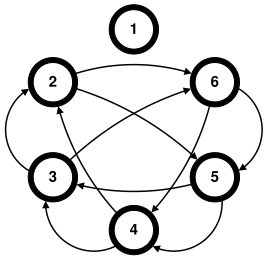

Figure 3. Sponsor-graph 2 slots after the failure of node 1.

moment, before reconfiguration, there are edges directed to each node, therefore agreement is ensured.

3.6 Further Considerations

A node is expected to broadcast not only its message but also the information necessary for membership agreement. For practical purposes the total order in the membership set can be adapted in order to permit a "reaction time" from processing nodes. If a node is to start the broadcast immediately after the previous slot ends, there may be insufficient time to evaluate the correctness of the previous message and to execute the membership protocol. At the expense of detection latency, each node N_i can be set to be sponsoring the nodes starting from N_{i-2} instead of sponsoring its immediate predecessor.

Consider a bus with a total bandwidth of 1Mbps where each round takes 5ms to complete. There are 5Kbits available in each round. In its minimum configuration, the protocol requires 3 bits piggybacked on each message (assuming that access to an event-driven schedule is not available). If we assume that the system contains 10 processing nodes, then there will be an overhead of 30 bits in each round, dedicated to the membership service. This corresponds to less than 1% of the total bandwidth. A noteworthy fact is that the overhead grows linearly (with a factor of $k+1$) with the number of nodes.

Regarding the reintegration procedure, if we consider non-negligible the probability of i) two nodes attempting reintegration in the same round, ii) an inconsistent failure of the second node occurs during the first reintegration attempt and iii) there is another inconsistent failure in the system, then the fault-tolerance of the reintegration process has to be achieved by tentative reintegration messages. Along with the reintegrating variable each node should keep track of the number of tentative reintegrations. The decision about the reintegration of a node will only be made after a predefined number of slots.

If, on the other hand, we wish to extend the functionality of the reintegration process by allowing the simultaneous reintegration of multiple nodes, the number of overhead $r\text{-}ack$ bits can be increased (by using a counter of currently reintegrating nodes).

4 Related Research

The protocol specification allows a comparison with existing membership protocols. The TTP specification [16] includes a membership service which imposes no direct overhead on the network. This is achieved instead by placing the load of the failure detection on the CRC mechanisms and by requiring the membership state to be explicitly broadcasted in order to allow the reintegration of previously failed nodes. In our approach a node needs only to listen on the bus for the period of one round on the bus to recover the membership state.

Ref. [14] presents a solution which isolates TTP's membership protocol from the other elements in the protocol. By using a single acknowledgement bit it ensures membership agreement, under the single failure assumption, by prompt removal of failed nodes. Since the proposed scheme does not provide reintegration capabilities, prompt removal is a costly approach to tolerating transient failures. Nevertheless, this scheme requires only one overhead bit per message, whereas our approach imposes an overhead of at least two acknowledgement bits to tolerate any random failure.

In [12] a solution based on a variable number of sponsors is presented, but in a context unrelated to hard real-time systems. Ref. [17] briefly and informally presents a protocol based on a variable number of sponsors as well. In the proposed scheme, permanent node failures will lead to a rapid decrease in the reliability of the protocol. On the other hand, reintegrating nodes is easy and does not incur additional overhead.

5 Application Process Group Membership through Fail-Reporting

Today many manufacturers of embedded systems are facing reliability and cost problems because their products are composed of an increasing number of subsystems and functions, which each use their own microcontroller. To address these problems, there are currently several initiatives under way aiming at simplifying the sharing of computing resources among different functions in distributed embedded real-time systems. The development of the ARINC 653 [1] standard for the aerospace industry and the AUTOSAR [13] project launched by the automotive industry are examples of such initiatives.

One goal of these initiatives is to simplify the construction of distributed systems where many functions share a

IEEE COMPUTER SOCIETY

common hardware platform with relatively few but powerful processing elements. Such designs have a great potential to reduce both product and maintenance costs, and improve system reliability, since they require fewer hardware components than designs where every function has their own dedicated microcontroller. However, to achieve these improvements, it is necessary to equip the system with powerful mechanisms for ensuring *node-level fault containment* between different application processes, and between the application processes and the operating system. These systems require powerful microcontrollers with hardware implemented memory protection mechanisms. Freescale's MPC5554 [10] is an example of a recently introduced microcontroller that fulfills these requirements.

From a fault-tolerance perspective there are clear differences between systems that do not provide memory protection and those that do. In the former, the smallest unit of failure typically corresponds to an entire node, while in the later the smallest unit of failure is an application process. Thus, for systems provided with memory protection and other mechanisms for achieving node-level fault containment, the membership service should preferably handle both node failures and application process failures.

So far, we have described our protocol in terms of a processor group membership protocol. However, it can easily be extended to keep track of both node failures and application process failures. Without giving a full protocol description, we will here outline how such an extension could be implemented.

We assume that a node equipped with mechanisms for node-level fault containment can exhibit two different failure modes: *fail-silent* and *fail-reporting*. A fail-silent failure implies that the node does not produce any results at all. We assume that fail-silent failures only occur when the entire node has failed, for example, as a result of an operating system crash or a permanent hardware failure.

On the other hand, a fail-reporting failure occurs when a node cannot send a result because an application process has failed. In that case, the operating system is still operational and can produce a failure report message that the node sends instead of the regular message. Such failure reports could of course be generated without the involvement of the processor group membership protocol. However, we believe that there are many advantages to combine the processor group and application process group membership into a single service. To achieve this for our protocol, we can simply add one extra bit along with the k acknowledgement bits and the reintegration bit to our message format. This fail-report bit would indicate that a message is carrying a failure report rather than a regular message. When multiple applications (running in the same node) share the same bus-slot to send their messages one bit is added for each application, with the same indication.

The payload part of the message could optionally carry a failure code that could be passed on to application processes which cooperate with the failed process. This way the membership service could be used to support application specific recovery or error handling actions. The failure codes could be generated by either the operating system or the failing processes itself, or possibly by a specially designed network controller.

With respect to the processor group membership protocol, there is no difference between messages that carry failure reports and those that carry regular messages. That is, the acknowledgment bits and reconfiguration bit will have the same function for both message types. Similarly, a message containing a failure report will be handled by the node membership service exactly in the same way as a regular message.

6 Conclusions and Future Work

This paper provided the precise specification of a protocol for membership agreement in synchronous systems. This protocol is flexible in the sense that the trade-off between reliability and communication overhead can be defined at design time. Additionally, agreement on departure is separated from agreement on reintegration. This feature allows the cost-effective implementation of reliable reintegration by using the event-driven schedule, instead of the time-triggered part, in buses such as FlexRay or TTCAN.

The existence of structural redundancy (e.g. duplicate buses) supports our assumptions on the failure model. However, the protocol is designed to provide a fair level of reliability even when there is no redundancy. In networks based on a bus topology or a star topology, a transient failure is likely to affect either all receiving nodes or a single node. When designing mission-critical systems it is possible to increase the fault-tolerance of the protocol by specifying the failure rate that is to be endured and setting the necessary overhead accordingly.

Reaching agreement is based on the principle that a node should not be removed from the membership in the event of a transient failure. Following this principle implies that the layers built on top of the membership, in particular the applications, are able to handle omission failures. This arises from the fact that single transient receiving failures result in lost messages without membership departures. It is possible to extend the protocol with a configuration parameter that determines whether or not a node should voluntarily depart the membership when a message is missed but the evidence of its reception by other nodes is received. Such an approach would make it possible to use the protocol in applications which do not allow omission failures. This extension to the protocol is left for future work.

In addition to node membership, this paper specifies how

IEEE COMPUTER SOCIETY

application membership may be provided. Such a service is intended for systems which benefit from the knowledge about the operational state of individual applications. We base application membership on the fail-reporting behavior – an extension of the fail-silent behavior by producing either the correct result or a report on the cause of failure.

Throughout this paper we have limited the discussion to systems where each node has a single slot dedicated in each round. When building systems where nodes run several different applications it might be useful for a node to broadcast more than once per communication round. It would therefore be interesting to overcome this limitation, while taking into account that a node should not acknowledge its own messages when specifying the sponsoring relations.

Any protocol devised for membership agreement must be formally verified before it can be made part of a fault-tolerant computer system. One of the next steps is therefore to formally verify the correctness of the protocol by using, for instance, a model-checking tool.

However, formal methods evaluate the correctness of any implementation by taking into account all possibilities, including infrequent worst-case failure-scenarios. Thus, it would be also interesting to study the reliability of the protocol from a stochastic point of view. Such a study would take into consideration the failure rates to measure the reliability of the protocol.

References

[1] ARINC Incorporated. ARINC specification 653-1: Avionics application software standard interface, 2006.

[2] A. Avižienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[3] J. Berwanger, C. Ebner, A. Schedl, R. Belschner, S. Fluhrer, P. Lohrmann, E. Fuchs, D. Millinger, M. Sprachmann, F. Bogenberger, G. Hay, A. Krüger, M. Rausch, W. Budde, P. Fuhrmann, and R. Mores. FlexRay: The communication system for advanced automotive control systems. *SAE transactions*, 110(7):303–314, 2001.

[4] V. Claesson, H. Lönn, and N. Suri. An efficient TDMA start-up and restart synchronization approach for distributed embedded systems. *IEEE Trans. Parallel Distrib. Syst.*, 15(8):725–739, 2004.

[5] M. Clegg and K. Marzullo. A low-cost processor group membership protocol for a hard real-time distributed system. In *IEEE Real-Time Systems Symposium*, pages 90–98. IEEE Computer Society, 1997.

[6] F. Cristian. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing*, 4:175–187, 1991.

[7] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Parallel and Distrib. Systems*, 10(6):642–657, 1999.

[8] P. D. Ezhilchelvan, R. A. Macêdo, and S. K. Shrivastava. Newtop: A fault-tolerant group communication protocol. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95)*, pages 296–306, Los Alamitos, CA, USA, May 30–June 2 1995.

[9] C. Fetzer and F. Christian. A fail-aware membership service. In *Proceedings of The 16th Symposium on Reliable Distributed Systems (SRDS '97)*, pages 157–164, Oct. 1997.

[10] Freescale Semiconductor, Inc. *MPC5554 Product Summary Page, http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC5554*, September 15, 2006.

[11] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN (Time Triggered CAN - TTCAN). Technical report, Robert Bosch GmbH, 2000.

[12] R. Golding. *Weak-Consistency Group Communication and Membership*. PhD thesis, University of California, Santa Cruz, USA, 1992.

[13] H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Mat, K. Nishikawa, and T. Scharnhorst. Automotive open system architecture - an industry-wide initiative to manage the complexity of emerging automotive e/e architectures. In *Proc. Convergence*, 2004.

[14] S. Katz, P. Lincoln, and J. Rushby. Low-overhead time-triggered group membership. *Lecture Notes in Computer Science*, 1320:155–169, 1997.

[15] K. H. Kim, H. Kopetz, K. Mori, E. Shokri, and G. Grünsteidl. An efficient decentralized approach to processor-group membership maintenance in real-time LAN systems: The PRHB/ED scheme. In *Symposium on Reliable Distributed Systems*, pages 74–83, 1992.

[16] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[17] H. Lönn. *Synchronization and Communication Results in Safety-Critical Real-Time Systems*. PhD thesis, Department of Computer Engineering, Chalmers University of Technology, Gothenburg, Sweden, 1999.

[18] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. Processor membership in asynchronous distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 5(5):459–473, 1994.

[19] A. Ricciardi and K. P. Birman. Using process groups to implement failure detection in asynchronous environments. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 341–353, Montreal, Quebec, Canada, 19–21 Aug. 1991.

[20] L. Rodrigues, P. Veríssimo, and J. Rufino. A low-level processor group membership protocol for LANS. In *ICDCS*, pages 541–550, 1993.

[21] J. M. Rushby. Bus architectures for safety-critical embedded systems. In T. A. Henzinger and C. M. Kirsch, editors, *Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323. Springer, 2001.

0-7695-2724-8/06 $20.00 © 2006 IEEE

IEEE COMPUTER SOCIETY