Invited paper, presented at FTRTFT'02, Oldenburg, Germany, September 2002. Springer-Verlag LNCS Vol. 2469, pp. 83–105.

An Overview of Formal Verification For the Time-Triggered Architecture*

John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025, USA
rushby@csl.sri.com

Abstract. We describe formal verification of some of the key algorithms in the Time-Triggered Architecture (TTA) for real-time safety-critical control applications. Some of these algorithms pose formidable challenges to current techniques and have been formally verified only in simplified form or under restricted fault assumptions. We describe what has been done and what remains to be done and indicate some directions that seem promising for the remaining cases and for increasing the automation that can be applied. We also describe the larger challenges posed by formal verification of the interaction of the constituent algorithms and of their emergent properties.

1 Introduction

The Time-Triggered Architecture (TTA) provides an infrastructure for safety-critical real-time control systems of the kind used in modern cars and airplanes. Concretely, it comprises an interlocking suite of distributed algorithms for functions such as clock synchronization and group membership, and their implementation in the form of TTA controllers, buses, and hubs. The suite of algorithms is known as TTP/C (an adjunct for non safety-critical applications is known as TTP/A) and was originally developed by Kopetz and colleagues at the Technical University of Vienna [28]; its current specification and commercial realization are by TTTech of Vienna [75]. More abstractly, TTA is part of a comprehensive approach to safety-critical real-time system design [25] that centers on time-triggered operation [26] and includes notions such as "temporal firewalls" [24] and "elementary" interfaces [27].

The algorithms of TTA are an exciting target for formal verification because they are individually challenging and they interact in interesting ways. To practitioners and developers of formal verification methods and their tools, these algorithms are excellent test cases—first, to be able to verify them at all, then to be able to verify them with sufficient automation that the techniques used can plausibly be transferred to nonspecialists

* This research was supported by NASA Langley Research Center under Cooperative Agreement NCC-1-377 with Honeywell Incorporated, by DARPA through the US Air Force Rome Laboratory under Contract F30602-96-C-0291, by the National Science Foundation under Contract CCR-00-86096, and by the NextTTA project of the European Union.

for use in similar applications. For the developers and users of TTA, formal verification provides valuable assurance for its safety-critical claims, and explication of the assumptions on which these rest. As new versions of TTA and its implementations are developed, there is the additional opportunity to employ formal methods in the design loop.

TTA provides the functionality of a bus: host computers attach to TTA and are able to exchange messages with other hosts; in addition, TTA provides certain services to the hosts (e.g., an indication which other hosts and their interface controllers are participating reliably in network protocols). Because it is used in safety-critical systems, TTA must be fault tolerant: that is, it must continue to provide its services to nonfaulty hosts in the presence of faulty hosts and in the presence of faults in its own components. In addition, the services that it provides to hosts are chosen to ease the design and construction of fault-tolerant applications (e.g., in an automobile brake-by-wire application, each wheel has a brake that is controlled by its own host computer; the services provided by TTA make it fairly simple to arrange a safe distributed algorithm in which each host can adjust the braking force applied to its wheel to compensate for the failure of one of the other brakes or its host).

Serious consideration of fault-tolerant systems requires careful identification of the fault containment units (components that fail independently), fault hypotheses (the kind, arrival rate, and total number of faults to be tolerated), and the type of fault tolerance to be provided (e.g., what constitutes acceptable behavior in the presence of faults: fault masking vs. fail silence, self stabilization, or never-give-up). The basic goal in verifying a fault-tolerant algorithm is to prove

fault hypotheses satisfied *implies* acceptable behavior.

Stochastic or other probabilistic and experimental methods must then establish that the probability of the fault hypotheses being satisfied is sufficiently large to satisfy the mission requirements.

In this short paper, it is not possible to provide much by way of background to the topics adumbrated above, nor to discuss the design choices in TTA, but a suitable introduction is available in a previous paper [54] (and in more detail in [55]). Neither is it possible, within the limitations of this paper, to describe in detail the formal verifications that have already been performed for certain TTA algorithms. Instead, my goal here is to provide an overview of these verifications, and some of their historical antecedents, focusing on the importance of the exact fault hypotheses that are considered for each algorithm and on the ways in which the different algorithms interact. I also indicate techniques that increase the amount of automation that can be used in these verifications, and suggest approaches that may be useful in tackling some of the challenges that still remain.

2 Clock Synchronization

As its full name indicates, the Time-Triggered Architecture uses the passage of time to schedule its activity and to coordinate its distributed components. A fault tolerant distributed clock synchronization algorithm is therefore one of TTA's fundamental elements.

Host computers attach to TTA through an interface controller that implements the TTP/C protocol. I refer to the combination of a host and its TTA controller as a *node*. Each controller contains an oscillator from which it derives its local notion of time (i.e., a clock). Operation of TTA is driven by a global schedule, so it is important that the local clocks are always in close agreement. Drift in the oscillators causes the various local clocks to drift apart so periodically (several hundred times a second) they must be resynchronized. What makes this difficult is that some of the clocks may be faulty.

The clock synchronization algorithm used in TTA is a modification of the Welch-Lynch (also known as Lundelius-Lynch) algorithm [78], which itself can be understood as a particular case of the abstract algorithm described by Schneider [66]. Schneider's abstract algorithm operates as follows: periodically, the nodes decide that it is time to resynchronize their clocks, each node determines the skews between its own clock and those of other nodes, forms a *fault-tolerant average* of these values, and adjusts its own clock by that amount.

An intuitive explanation for the general approach is the following. After a resynchronization, all the nonfaulty clocks will be close together (this is the definition of synchronization); by the time that they next synchronize, the nonfaulty clocks may have drifted further apart, but the amount of drift is bounded (this is the definition of a good clock); the clocks can be brought back together by setting them to some value close to the middle of their spread. An "ordinary average" (e.g., the mean or median) over all clocks may be affected by wild readings from faulty clocks (which, under a *Byzantine* fault hypothesis, may provide different readings to different observers), so we need a "fault-tolerant average" that is insensitive to a certain number of readings from faulty clocks.

The Welch-Lynch algorithm is characterized by use of the *fault-tolerant midpoint* as its averaging function. If we have n clocks and the maximum number of simultaneous faults to be tolerated is k ($3k < n$), then the fault-tolerant midpoint is the average of the $k + 1$'st and $n - k$'th clock skew readings, when these are arranged in order from smallest to largest. If there are at most k faulty clocks, then some reading from a nonfaulty clock must be at least as small as the $k + 1$'st reading, and the reading from another nonfaulty clock must be at least as great as the $n - k$'th; hence, the average of these two readings should be close to the middle of the spread of readings from good clocks.

The TTA algorithm is basically the Welch-Lynch algorithm specialized for $k = 1$ (i.e., it tolerates a single fault): that is, clocks are set to the average of the 2nd and $n - 1$'st clock readings (i.e., the second-smallest and second-largest). This algorithm works and tolerates a single arbitrary fault whenever $n \geq 4$. TTA does not use dedicated wires to communicate clock readings among the nodes attached to the network; instead, it exploits the fact that communication is time triggered according to a global schedule. When a node a receives a message from a node b, it notes the reading of its local clock and subtracts a fixed correction term to account for the network delay; the difference between this adjusted clock reading and the time for b's transmission that is indicated in the global schedule yields a's perception of the skew between clocks a and b.

Not all nodes in a TTA system need have accurate oscillators (they are expensive), so TTA's algorithm is modified from Welch-Lynch to use only the clock skews from nodes marked[1] as having accurate oscillators. Analysis and verification of this variant can be adapted straightforwardly from that of the basic algorithm. Unfortunately, TTA adds another complication.

For scalability, an implementation on the Welch-Lynch algorithm should use data structures that are independent of the number of nodes—i.e., it should not be necessary for each node to store the clock difference readings for all (accurate) clocks. Clearly, the second-smallest clock difference reading can be determined with just two registers (one to hold the smallest and another for the second-smallest reading seen so far), and the second-largest can be determined similarly, for a total of four registers per node. If TTA used this approach, verification of its clock synchronization algorithm would follow straightforwardly from that of Welch-Lynch. Instead, for reasons that are not described, TTA does not consider all the accurate clocks when choosing the second-smallest and second-largest, but just four of them.

The four clocks considered for synchronization are chosen as follows. First, TTA is able to tolerate more than a single fault by reconfiguring to exclude nodes that are detected to be faulty. This is accomplished by the group membership algorithm of TTA, which is discussed in the following section.[2] The four clocks considered for synchronization are chosen from the members of the current membership; it is therefore essential that group membership have the property that all nonfaulty nodes have the same members at all times. Next, each node maintains a queue of four clock readings[3]; whenever a message is received from a node that is in the current membership and that has the `SYF` field set, the clock difference reading is pushed on to the receiving node's queue (ejecting the oldest reading in the queue). Finally, when the current slot has the synchronization field (`CS`) set in the `MEDL`, each node runs the synchronization algorithm using the four clock readings stored in its queue.

Formal verification of the TTA algorithm requires more than simply verifying a four-clocks version of the basic Welch-Lynch algorithm: for example, the chosen clocks can change from one round to the next. However, verification of the basic algorithm provides a foundation for the TTA case.

Formal verification of clock synchronization algorithms has quite a long history, beginning with Rushby and von Henke's verification [60] of the interactive convergence algorithm of Lamport and Melliar Smith [32]; this is similar to the Welch-Lynch algorithm, except that the *egocentric mean* is used as the fault-tolerant average. Shankar [70] formally verified Schneider's abstract algorithm and its instantiation for interactive convergence. This formalization was subsequently improved by Miner (reducing the difficulty of the proof obligations needed to establish the correctness of specific instantiations), who also verified the Welch-Lynch instantiation [38]. All these verifications were undertaken with EHDM [61], a precursor to PVS [41]. The treatment developed by

[1] By having the `SYF` field set in the `MEDL` (the global schedule known to all nodes).

[2] A node whose clock loses synchronization will suffer send and/or receive faults and will therefore be detected and excluded by the group membership algorithm.

[3] It is described as a push-down stack in the TTP/C specification [75], but this seems to be an error.

Miner was translated to PVS and generalized (to admit nonaveraging algorithms such as that of Srikanth and Toueg [73] that do not conform to Schneider's treatment) by Schwier and von Henke [69]. This treatment was then extended to the TTA algorithm by Pfeifer, Schwier and von Henke [45]. The TTA algorithm is intended to operate in networks where there are at least four good clocks, and it is able to mask any single fault in this circumstance. Pfeifer, Schwier and von Henke's verification establishes this property. Additional challenges still remain, however.

In keeping with the *never give up* philosophy that is appropriate for safety-critical applications, TTA should remain operational with less than four good clocks, though "the requirement to handle a Byzantine fault is waived" [75, page 85]. It would be valuable to characterize and formally verify the exact fault tolerance achieved in these cases. One approach to achieving this would be to undertake the verification in the context of a "hybrid" fault model such as that introduced for consensus by Thambidurai and Park [74]. In a pure Byzantine fault model, all faults are treated as arbitrary: nothing is assumed about the behavior of faulty components. A hybrid fault model introduces additional, constrained kinds of faults and the verification is extended to examine the behavior of the algorithm concerned under combinations of several faults of different kinds. Thambidurai and Park's model augments the Byzantine or *arbitrary* fault model with *manifest* and *symmetric* faults. A manifest fault is one that is consistently detectable by all nonfaulty nodes; a symmetric fault is unconstrained, except that it appears the same to all nonfaulty nodes. Rushby reinterpreted this fault model for clock synchronization and extended verification of the interactive convergence algorithm to this more elaborate fault model [49]. He showed that the interactive convergence algorithm with n nodes can withstand a arbitrary, s symmetric, and m manifest faults simultaneously, provided $n > 3a + 2s + m$. Thus, a three-clock system using this algorithm can withstand a symmetric fault or two manifest faults.

Rushby also extended this analysis to *link* faults, which can be considered as asymmetric and possibly intermittent manifest faults (i.e., node a may obtain a correct reading of node b's clock while node c obtains a detectably faulty reading). The fault tolerance of the algorithm is then $n > 3a + 2s + m + l$ where l is the maximum, over all pairs of nodes, of the number of nodes that have faulty links to one or other of the pair.

It would be interesting to extend formal verification of the TTA algorithm to this fault model. Not only would this enlarge the analysis to cases where fewer than three good clocks remain, but it could also provide a much simpler way to deal with the peculiarities of the TTA algorithm (i.e., its use of queues of just four clocks). Instead of explicitly modeling properties of the queues, we could, under a fault model that admits link faults, imagine that the queues are larger and contain clock difference readings from the full set of nodes, but that link faults reduce the number of valid readings actually present in each queue to four (this idea was suggested by Holger Pfeifer). A recent paper by Schmid [64] considers link faults for clock synchronization in a very general setting, and establishes bounds on fault tolerance for both the Welch-Lynch and Srikanth-Toueg algorithms and I believe this would be an excellent foundation for a comprehensive verification of the TTA algorithm.

All the formal verifications of clock synchronizations mentioned above are "brute force": they are essentially mechanized reproductions of proofs originally undertaken by hand. The proofs depend heavily on arithmetic reasoning and can be formalized at reasonable cost only with the aid of verification systems that provide effective mechanization for arithmetic, such as PVS. Even these systems, however, typically mechanize only linear arithmetic and require tediously many human-directed proof steps (or numerous intermediate lemmas) to verify the formulas that arise in clock synchronization. The new ICS decision procedures [16] developed for PVS include (incomplete) extensions to nonlinear products and it will be interesting to explore the extent to which such extensions simplify formal verification of clock synchronization algorithms.[4] Even if all the arithmetic reasoning were completely automated, current approaches to formal verification of clock synchronization algorithms still depend heavily on human insight and guidance. The problem is that the synchronization property is not inductive: it must be strengthened by the conjunction of several other properties to achieve a property that is inductive. These additional properties are intricate arithmetic statements whose invention seems to require considerable human insight. It would be interesting to see if modern methods for invariant discovery and strengthening [6, 7, 76] can generate some of these automatically, or if the need for them could be sidestepped using reachability analysis on linear hybrid automata.

All the verifications described above deal with the steady-state case; initial synchronization is quite a different challenge. Note that (re)initialization may be required during operation if the system suffers a massive failure (e.g., due to powerful electromagnetic effects), so it must be fast. The basic idea is that a node that detects no activity on the bus for some time will assume that initialization is required and it will broadcast a wakeup message: nodes that receive the message will synchronize to it. Of course, other nodes may make the same determination at about the same time and may send wakeup messages that collide with others. In these cases, nodes back off for (different) node-specific intervals and try again. However, it is difficult to detect collisions with perfect accuracy and simple algorithms can lead to existence of groups of nodes synchronized within themselves but unaware of the existence of the other groups. All of these complications must be addressed in a context where some nodes are faulty and may not be following (indeed, may be actively disrupting) the intended algorithm. The latest version of TTA uses a star topology and the initialization algorithm is being revised to exploit some additional safeguards that the central guardian makes possible [42]. Verification of initialization algorithms is challenging because, as clearly explained in [42], the essential purpose of such an algorithm is to cause a transition between two models of computation: from asynchronous to synchronous. Formal explication of this issue, and verification of the TTA initialization algorithm, are worthwhile endeavors for the future.

3 Transmission Window Timing

Synchronized clocks and a global schedule ensure that nonfaulty nodes broadcast their messages in disjoint time slots: messages sent by nonfaulty nodes are guaranteed not

[4] It is not enough to mechanize real arithmetic on its own; it must be combined with inequalities, integer linear arithmetic, equality over uninterpreted function symbols and several other theories [50].

to collide on the bus. A faulty node, however, could broadcast at any time—it could even broadcast constantly (the *babbling* failure mode). This fault is countered by use of a separate fault containment unit called a *guardian* that has independent knowledge of the time and the schedule: a message sent by one node will reach others only if the guardian agrees that it is indeed scheduled for that time.

Now, the sending node, the guardian, and each receiving node have synchronized clocks, but there must be some slack in the time window they assign to each slot so that good messages are not truncated or rejected due to clock skew within the bounds guaranteed by the synchronization algorithm. The design rules used in TTA are the following, where Π is the maximum clock skew between synchronized components.

- The receive window extends from the beginning of the slot to $4\,\Pi$ beyond its allotted duration.
- Transmission begins $2\,\Pi$ units after the beginning of the slot and should last no longer than the allotted duration.
- The bus guardian for a transmitter opens its window Π units after the beginning of the slot and closes it $3\,\Pi$ beyond its allotted duration.

These rules are intended to ensure the following requirements.

Agreement: If any nonfaulty node accepts a transmission, then all nonfaulty nodes do.
Validity: If any nonfaulty node transmits a message, then all nonfaulty nodes will accept the transmission.
Separation: messages sent by nonfaulty nodes or passed by nonfaulty guardians do not arrive before other components have finished the previous slot, nor after they have started the following one.

Formal specification and verification of these properties is a relatively straightforward exercise. Description of a formal treatment using PVS is available as a technical report [57].

4 Group Membership

The clock synchronization algorithm tolerates only a single (arbitrary) fault. Additional faults are tolerated by diagnosing the faulty node and reconfiguring to exclude it. This diagnosis and reconfiguration is performed by the *group membership* algorithm of TTA, which ensures that each TTA node has a record of which nodes are currently participating correctly in the TTP/C protocol. In addition to supporting the internal fault tolerance of TTA, membership information is made available as a service to applications; this supports the construction of relatively simple, but correct, strategies for tolerating faults at the application level. For example, in an automobile brake-by-wire application, the node at each wheel can adjust its braking force to compensate for the failure (as indicated in the membership information) of the node or brake at another wheel. For such strategies to work, it is obviously necessary that the membership information should be reliable, and that the application state of nonmembers should be predictable (e.g., the brake is fully released).

Group membership is a distributed algorithm: each node maintains a private *membership* list, which records all the nodes that it believes to be nonfaulty. Reliability of the membership information is characterized by the following requirements.

Agreement: The membership lists of all nonfaulty nodes are the same.

Validity: The membership lists of all nonfaulty nodes contain all nonfaulty nodes and at most one faulty node (we cannot require immediate removal of faulty nodes because a fault must be manifested before it can be diagnosed).

These requirements can be satisfied only under restricted fault hypotheses. For example, validity cannot be satisfied if new faults arrive too rapidly, and it is provably impossible to diagnose an arbitrary-faulty node with certainty. When unable to maintain accurate membership, the best recourse is to maintain agreement, but sacrifice validity. This weakened requirement is called *clique avoidance*.

Two additional properties also are desirable in a group membership algorithm.

Self-diagnosis: faulty nodes eventually remove themselves from their own membership lists and fail silently (i.e., cease broadcasting).

Reintegration: it should be possible for excluded but recovered nodes to determine the current membership and be readmitted.

TTA operates as a broadcast bus (even though the recent versions are stars topologically); the global schedule executes as a repetitive series of *rounds*, and each node is allocated a broadcast slot in each round. The fault hypothesis of the membership algorithm is a benign one: faults must arrive two or more rounds apart, and must be symmetric in their manifestations: either *all* or exactly *one* node may fail to receive a broadcast message (the former is called a *send* fault, the latter a *receive* fault). The membership requirements would be relatively easy to satisfy if each node were to attach a copy of its membership list to each message that it broadcasts. Unfortunately, since messages are typically very short, this would use rather a lot of bandwidth (and bandwidth was a precious commodity in early implementations of TTA), so the algorithm must operate with less explicit information and nodes must infer the state and membership of other nodes through indirect means. This operates as follows.

Each active TTA node maintains a membership list of those nodes (including itself) that it believes to be active and operating correctly. Each node listens for messages from other nodes and updates its membership list according to the information that it receives. The time-triggered nature of the protocol means that each node knows when to expect a message from another node, and it can therefore detect the absence of such a message. Each message carries a CRC checksum that encodes information about its sender's *C-State*, which includes its local membership list. To infer the local membership of the sender of a message, receivers must append their estimate of that membership (and other C-state information) to the message and then check whether the calculated CRC matches that sent with the message. It is not feasible (or reliable) to try all possible memberships, so receivers perform the check against just their own local membership, and one or two variants.

Transmission faults are detected as follows: each broadcaster listens for the message from its *first successor* (roughly speaking, this will be the next node to broadcast) to check whether it suffered a transmission fault: this will be indicated by its exclusion from the membership list of the message from its first successor. However, this indication is ambiguous: it could be the result of a transmission fault by the original broadcaster, or of a receive fault by the successor. Nodes use the local membership carried by the message from their *second successor* to resolve this ambiguity: a membership that excludes the original broadcaster but includes the first successor indicates a transmission fault by the original broadcaster, and one that includes the original broadcaster but excludes the first successor indicates a receive fault by the first successor.

Nodes that suffer receive faults could diagnose themselves in a similar way: their local membership lists will differ from those of nonfaulty nodes, so their next broadcast will be rejected by both their successors. However, the algorithm actually performs this diagnosis differently. Each node maintains *accept* and *reject* counters that are initialized to 1 and 0, respectively, following its own broadcast. Incoming messages that indicate a membership matching that of the receiver cause the receiver to increment its accept count; others (i.e., those that indicate a different membership or that are considered invalid for other reasons) cause it to increment its reject count. Before broadcasting, each node compares its accept and reject counts and shuts down unless the former is greater than the latter.

Formal verification of this algorithm is difficult. We wish to prove that agreement and validity are invariants of the algorithm (i.e., they are true of all reachable states), but it is difficult to do this directly (because it is hard to characterize the reachable states). So, instead, we try to prove a stronger property: namely, that agreement and validity are *inductive* (that is, true of the initial states and preserved by all steps of the algorithm). The general problem with this approach to verification of safety properties of distributed algorithms is that natural statements of the properties of interest are seldom inductive. Instead, it is necessary to strengthen them by conjoining additional properties until they become inductive. The additional properties typically are discovered by examining failed proofs and require human insight.

Before details of the TTA group membership algorithm were known, Katz, Lincoln, and Rushby published a different algorithm for a similar problem, together with an informal proof of its correctness [23] (I will call this the "WDAG" algorithm). A flaw in this algorithm for the special case of three nodes was discovered independently by Shankar and by Creese and Roscoe [12] and considerable effort was expended in attempts to formally verify the corrected version. A suitable method was found by Rushby [53] who used it to formally verify the WDAG algorithm, but used a simplified algorithm (called the "CAV" algorithm) to explicate the method in [53]. The method is based on strengthening a putative safety property into a *disjunction* of "configurations" that can easily be proved to be inductive. Configurations can be constructed systematically and transitions among them have a natural diagrammatic representation that conveys insight into the operation of the algorithm. Pfeifer subsequently used this method to verify validity, agreement, and self-diagnosis for the full TTA membership algorithm [44] (verification of self-diagnosis is not described in the paper).

Although the method just described is systematic, it does require considerable human interaction and insight, so more automatic methods are desirable. All the group membership algorithms mentioned (CAV, WDAG, TTA) are n-process algorithms (so-called *parameterized systems*), so one attractive class of methods seeks to reduce the general case to some fixed configuration (say four processes) of an abstracted algorithm that can be model checked. Creese and Roscoe [12] report an investigation along these lines for the WDAG algorithm. The difficulty in such approaches is that proving that the abstracted algorithm is faithful to the original is often as hard as the direct proof.

An alternative is to *construct* the abstracted algorithm using automated theorem proving so that the result is guaranteed to be sound, but possibly too conservative. These methods are widely used for predicate [62] and data [11] abstraction (both methods are implemented in PVS using a generalization of the technique described in [63]), and have been applied to n-process examples [71]. The precision of an abstraction is determined by the guidance provided to the calculation (e.g., which predicates to abstract on) and by the power of the automated deduction methods that are employed.[5] The logic called WS1S is very attractive in this regard, because it is very expressive (it can represent arithmetic and set operations on integers) and it is decidable [14]. The method implemented in the PAX tool [4, 5] performs automated abstraction of parameterized specifications modeled in WS1S. Application of the tool to the CAV group membership protocol is described on the PAX web page at `http://www.informatik.uni-kiel.de/~kba/pax/examples.html`. The abstraction yields a finite-state system that can be examined by model checking. I conjecture that extension of this method to the TTA algorithm may prove difficult because the counters used in that algorithm add an extra unbounded dimension.

The design of TTA (and particularly of the central guardian) is intended to minimize violations of the benign fault hypothesis of the group membership algorithm. But we cannot guarantee absence of such violations, so the membership algorithm is buttressed by a clique avoidance algorithm (it would better be called a clique elimination algorithm) that sacrifices validity but maintains agreement under weakened fault hypotheses. Clique avoidance is actually a subalgorithm of the membership algorithm: it comprises just the part that manages the accept and reject counters and that causes a node to shut down prior to a broadcast unless its accept count exceeds its reject count at that point. The clique avoidance algorithm can be analyzed either in isolation or, more accurately, in the presence of the rest of the membership algorithm (this is, the part that deals with the first and second successor).

Beyond the benign fault hypothesis lie *asymmetric* faults (where more than one but less than all nodes fail to receive a broadcast correctly), and *multiple* faults, which are those that arrive less than two rounds apart. These hypotheses all concern loss of messages; additional hypotheses include *processor* faults, where nodes fail to follow the algorithm, and *transient* faults, where nodes have their state corrupted (e.g., by high-intensity radiation) but otherwise follow the algorithm correctly.

Bauer and Paulitsch [3] describe the clique avoidance algorithm and give an informal proof that it tolerates a single asymmetric fault. Their analysis includes the effects of the rest of the membership algorithm. Bouajjani and Merceron [8] prove that the clique avoidance algorithm, considered in isolation, tolerates multiple asymmetric

[5] In this context, automated deduction methods are used in a *failure-tolerant* manner, so that if the methods fail to prove a true theorem, the resulting abstraction will be sound, but more conservative than necessary.

faults; they also describe an abstraction for the n-node, k-faults parameterized case that yields a counter automaton. Reachability is decidable for this class of systems, and experiments are reported with two automated verifiers for the $k = 1$ case.

For transient faults, I conjecture that the most appropriate framework for analysis is that of self-stabilization [68]. An algorithm is said to be *self-stabilizing* if it converges to a stable "good" state starting from an arbitrary initial state. The arbitrary initial state can be one caused by an electromagnetic upset (e.g., that changes the values of the accept and reject counters), or by other faults outside the benign fault hypotheses.

An attractive treatment of self-stabilization is provided by the "Detectors and Correctors" theory of Arora and Kulkarni. The full theory [2, 31] is comprehensive and more than is needed for my purposes, so I present a simplified and slightly modified version that adapts the important insights of the original formulation to the problem at hand.

We assume some "base" algorithm M whose purpose is to maintain an invariant S: that is, if the (distributed) system starts in a state satisfying the predicate S, then execution of M will maintain that property. In our case, M is the TTA group membership algorithm, and S is the conjunction of the agreement and validity properties. M corresponds to what Arora and Kulkarni call the "fault-intolerant" program, but in our context it is actually a fault-tolerant algorithm in its own right. This aspect of the system's operation can be specified by the Hoare formula

$$\{S\}\ M||F\ \{S\}$$

where F is a "fault injector" that characterizes the fault hypothesis of the base algorithm and $M||F$ denotes the concurrent execution of M and F.

Now, a transient fault can take the system to some state not satisfying S, and at this point our hope is that a "corrector" algorithm C will take over and somehow cause the system to converge to a state satisfying S, where the base algorithm can take over again. We can represent this by the following formula

$$C \models \Diamond S$$

where $\Diamond$ is the *eventually* modality of temporal logic.

In our case, C is the TTA clique avoidance algorithm. So far we have treated M and C separately but, as noted previously, they must actually run concurrently, so we really require

$$\{S\}\ C||M||F\ \{S\}$$

and

$$C||M||F \models \Diamond S.$$

The presence of F in the last of these represents the fact that although the disturbance that took the system to an arbitrary state is assumed to have passed when convergence begins, the standard, benign fault hypothesis still applies.

To ensure the first of these formulas, we need that C does not interfere with M—that is, that $C||M$ behaves the same as M (and hence $C||M||F$ behaves the same as $M||F$). A very direct way to ensure this is for C actually to be a subalgorithm of M—for then $C||M$ *is* the same as M. As we have already seen later, this is the case in TTA, where the clique avoidance algorithm is just a part of the membership algorithm.

A slight additional complication is that the corrector may not be able to restore the system to the ideal condition characterized by S, but only to some "safe" approximation to it, characterized by S'. This is the case in TTA, where clique avoidance sacrifices validity. Our formulas therefore become the following.

$$\{S\}\ C||M||F\ \{S\} \tag{1}$$

$$\{S'\}\ C||M||F\ \{S' \vee S\},\ \text{and}\ S \supset S' \tag{2}$$

and

$$C||M||F \models \Diamond S'. \tag{3}$$

The challenge is formally to verify these three formulas. Concretely, (1) is accomplished for TTA by Pfeifer's verification [44] (and potentially, in more automated form, by extensions to the approaches of [4, 8]), (2) should require little more than an adjustment to those proofs, and the hard case is (3). Bouajjani and Merceron's analysis [8] can be seen as establishing

$$C \models \Diamond S'$$

for the restricted case where the arbitrary initial state is one produced by the occurrence of multiple, possibly asymmetric faults in message transmission or reception. The general case must consider the possibility that the initial state is produced by some outside disturbance that sets the counters and flags of the algorithm to arbitrary values (I have formally verified this case for a simplified algorithm), and must also consider the presence of M and F. Formal verification of this general case is an interesting challenge for the future. Kulkarni [30, 31] has formally specified and verified the general detectors and correctors theory in PVS, and this provides a useful framework in which to develop the argument.

A separate topic is to examine the consequences of giving up validity in order to maintain agreement under the clique avoidance algorithm. Under the never give up philosophy, it is reasonable to sacrifice one property rather than lose all coordination when the standard fault hypothesis is violated, but some useful insight may be gained through an attempt to formally characterize the possible behaviors in these cases.

Reintegration has so far been absent from the discussion. A node that diagnoses a problem in its own operation will drop out of the membership, perform diagnostic tests and, if these are satisfactory (indicating that the original fault was a transient event), attempt to reintegrate itself into the running system. This requires that the node first (re)synchronizes its clock to the running system, then acquires the current membership, and then "speaks up" at its next slot in the schedule. There are potential difficulties here: for example, a broadcast by a node a may be missed by a node b whose membership is used to initialize a reintegrating node c; rejection of its message by b and c then causes the good node a to shut down. This scenario is excluded by the requirement that a reintegrating node must correctly receive a certain number of messages before it may broadcast itself. Formal examination of reintegration scenarios is another interesting challenge for the future.

5 Interaction of Clock Synchronization and Group Membership

Previous sections considered clock synchronization and group membership in isolation but noted that, in reality, they interact: synchronization depends on membership to eliminate nodes diagnosed as faulty, while membership depends on synchronization to create the time-triggered round structure on which its operation depends. Mutual dependence of components on the correct operation of each other is generally formalized in terms of assume-guarantee reasoning, first introduced by Chandy and Misra [39] and Jones [22]. The idea is to show that component X_1 guarantees certain properties P_1 on the assumption that component X_2 delivers certain properties P_2, and *vice versa* for X_2, and then claim that the composition of X_1 and X_2 guarantees P_1 and P_2 unconditionally. This kind of reasoning appears—and indeed is—circular in that X_1 depends on X_2 and *vice versa*. The circularity can lead to unsoundness and there has been much research on the formulation of rules for assume-guarantee reasoning that are both sound and useful. Different rules may be compared according to the kinds of system models and specification they support, the extent to which they lend themselves to mechanized analysis, and the extent to which they are preserved under refinement (i.e., the circumstances under which X_1 can be replaced by an implementation that may do more than X_1).

Closer examination of the circular dependency in TTA reveals that it is not circular if the temporal evolution of the system is taken into consideration: clock synchronization in round t depends on group membership in round $t-1$, which in turn depends on clock synchronization in round $t-2$ and so on. McMillan [37] has introduced an assume-guarantee rule that seems appropriate to this case. McMillan's rule can be expressed as follows, where H is a "helper" property (which can be simply *true*), $\Box$ is the "always" modality of Linear Temporal Logic (LTL), and $p \rhd q$ (" p constrains q") means that if p is always true up to time t, then q holds at time $t+1$ (i.e., p fails before q), where we interpret time as rounds.

$$\frac{\begin{array}{c}\langle H\rangle X_1 \langle P_2 \rhd P_1\rangle \\ \langle H\rangle X_2 \langle P_1 \rhd P_2\rangle\end{array}}{\langle H\rangle \;\; X_1 || X_2 \;\; \langle \Box(P_1 \wedge P_2)\rangle} \qquad (4)$$

Notice that $p \rhd q$ can be written as the LTL formula $\neg(p \,\mathsf{U}\, \neg q)$, where U is the LTL "until" operator. This means that the antecedent formulas can be established by LTL model checking if the transition relations for X_1 and X_2 are finite.

I believe the soundness of the circular interaction between the clock synchronization and group membership algorithms of TTA can be formally verified using McMillan's rule. To carry this out, we need to import the proof rule (4) into the verification framework employed—and for this we probably need to embed the semantics of the rule into the specification language concerned. McMillan's presentation of the rule only sketches the argument for its soundness; a more formal treatment is given by Namjoshi and Trefler [40], but it is not easy reading and does not convey the basic intuition. Rushby [56] presents an embedding of LTL in the PVS specification language and formally verifies the soundness of the rule. The specification and proof are surprisingly short and provide a good demonstration of the power and convenience of the PVS language and prover.

Using this foundation to verify the interaction between the clock synchronization and group membership algorithms of TTA remains a challenge for the future. Observe that such an application of assume-guarantee reasoning has rather an unusual character: conventionally, the components in assume-guarantee reasoning are viewed as separate, peer processes, whereas here they are distributed algorithms that form part of a protocol hierarchy (with membership above synchronization).

6 Emergent Properties

Clock synchronization, transmission window timing, and group membership are important properties, but what makes TTA useful are not the individual properties of its constituent algorithms, but the emergent properties that come about through their combination. These emergent properties are understood by the designers and advocates of TTA, but they have not been articulated formally in ways that are fully satisfactory, and I consider this the most important and interesting of the tasks that remain in the formal analysis of TTA.

I consider the three "top level" properties of TTA to be the time-triggered model of computation, support for application-independent fault tolerance, and partitioning. The time-triggered model of computation can be construed narrowly or broadly. Narrowly, it is a variant on the notion of synchronous system [35]: these are distributed computer systems where there are known upper bounds on the time that it takes nonfaulty processors to perform certain operations, and on the time that it takes for a message sent by one nonfaulty processor to be received by another. The existence of these bounds simplifies the development of fault-tolerant systems because nonfaulty processes executing a common algorithm can use the passage of time to predict each others' progress, and the absence of expected messages can be detected. This property contrasts with asynchronous systems, where there are no upper bounds on processing and message delays, and where it is therefore provably impossible to achieve certain forms of consistent knowledge or coordinated action in the presence of even simple faults [9, 17]. Rushby [52] presents a formal verification that a system possessing the synchronization and scheduling mechanisms of TTA can be used to create the abstraction of a synchronous system. An alternative model, closer to TTA in that it does not abstract out the real-time behavior, is that of the language Giotto [19] and it would be interesting to formalize the connection between TTA and Giotto.

More broadly construed, the notion of time-triggered system encompasses a whole philosophy of real-time systems design—notably that espoused by Kopetz [25]. Kopetz' broad conception includes a distinction between *composite* and *elementary* interfaces [27] and the notion of a *temporal firewall* [24].

A time-triggered system does not merely schedule activity within nodes, it also manages the reliable transmission of messages between them. Messages obviously communicate data between nodes (and the processes within them) but they may also, through their presence or absence and through the data that they convey, influence the flow of control within a node or process (or, more generically, a component). An important insight is that one component should not allow another to control its own progress. Suppose, for example, that the guarantees delivered by component X_1 are quite weak, such as, "this buffer may sometimes contain recent data concerning parameter A." Another component X_2 that uses this data must be prepared to operate when recent data about A is unavailable (at least from X_1). It might seem that predictability and simplicity would be enhanced if we were to ensure that the flow of data about A is reliable—perhaps using a protocol involving acknowledgments. But in fact, contrary to this intuition, such a mechanism would greatly increase the coupling between components and introduce more complicated failure propagations. For example, X_1 could block waiting for an acknowledgment from X_2 that may never come if X_2 has failed, thereby propagating the failure from X_2 to X_1. Kopetz [27] defines interfaces that involve such bidirectional flow of control as composite and argues convincingly that they should be eschewed in favor of elementary interfaces in which control flow is unidirectional.

The need for elementary interfaces leads to protocols for nonblocking asynchronous communication that nonetheless ensure timely transmission and mutual exclusion (i.e., no simultaneous reading and writing of the same buffer). In computer science, these are known as lock- and wait-free atomic register constructions ([1] is a convenient survey, focussing on the work of Lamport, who first introduced the topic), but similar constructions were developed independently in the avionics and real-time communities. The best-known of these is the four-slot protocol of Simpson [72]. Formal analyses of Simpson's protocol have been developed by Clark [10] (using Petri nets), by Rushby [59] (using model checking), and by Henderson and Paynter [18] (using PVS). Hesselink [21] have verified some atomic register constructions from the computer science literature using ACL2.

TTA uses a protocol called NBW (nonblocking write) [29] whose wait-free element was inspired by Simpson's algorithm, and whose lock-free construction is that of Lamport [33]. It would be useful to undertake a formal examination of NBW (which is used in the Communication Network Interface (CNI) that provides communication between hosts and their TTA controllers), particularly since Simpson's algorithm requires atomic control registers, and Rushby's analysis [59] shows that it fails when this (very strong) assumption is violated.

The larger issue of formally characterizing composite and elementary interfaces has not yet been tackled, to my knowledge. It is debatable whether formalization of these notions is best performed as part of a broad treatment of time-triggered systems, or as part of an orthogonal topic concerned with application-independent fault tolerance. *Temporal firewalls*, another element in Kopetz' comprehensive philosophy [24], seem definitely to belong in the treatment of fault tolerance. The standard way to communicate a sensor sample is to package it with a timestamp: then the consuming process can estimate the "freshness" of the sample. But surely the useful lifetime of a sample depends on the accuracy of the original reading and on the dynamics of the parameter being measured—and these factors are better known to the process doing the sensing than to the process that consumes the sample. So, argues Kopetz, it is better to turn the timestamp around, so that it indicates the "must use by" time, rather than the time at which the sample was taken. This is the idea of the temporal firewall, which exists in two variants. A *phase-insensitive* sensor sample is provided with a time and a guarantee that the sampled value is accurate (with respect to a specification published by the process that provides it) until the indicated time. For example, suppose that engine oil temperature may change by at most 1% of its range per second, that its sensor is completely accurate, and that the data is to be guaranteed to 0.5%. Then the sensor sample will be provided with a time 500 ms ahead of the instant when it was sampled, and the receiver will know that it is safe to use the sampled value until the indicated time. A *phase-sensitive* temporal firewall is used for rapidly changing parameters; in addition to sensor sample and time, it provides the parameters needed to perform state estimation. For example, along with sampled crankshaft angle, it may supply RPM, so that angle may be estimated more accurately at the time of use.

The advantage of temporal firewalls is that they allow some of the downstream processing (e.g., sensor fusion) to become less application dependent. Temporal firewalls are consistent with modern notions of *smart sensors* that co-locate computing resources with the sensor. Such resources allow a sensor to return additional information, including an estimate of the accuracy of its own reading. An attractive way to indicate (confidence in) the accuracy of a sensor reading is to return two values (both packaged in a temporal firewall) indicating the upper and lower 95% (say) confidence interval. If several such intervals are available from redundant sensors, then an interesting question is how best to combine (or *fuse*) them. Marzullo [36] introduces the sensor fusion function $\bigcap_{f,n}(S)$ for this problem; Rushby formally verifies the soundness of this construction (i.e., the fused interval always contains the correct value) [58]. A weakness of Marzullo's function is that it lacks the "Lipschitz Condition": small changes in input sensor readings can sometimes produce large changes in its output. Schmid and Schossmaier [65] have recently introduced an improved fusion function $\mathcal{F}_n^f(S)$ that does satisfy the Lipschitz condition, and is optimal among all such functions. It would be interesting to verify formally the properties of this function.

Principled fault tolerance requires not only that redundant sensor values are fused effectively, but that all redundant consumers agree on exactly the same values; this is the notion of *replica determinism* [46] that provides the foundation for *state machine replication* [67] and other methods for application-independent fault tolerance based on exact-match voting. Replica determinism in its turn depends on *interactively consistent* message passing: that is, message passing in which all nonfaulty recipients obtain the same value [43], even if the sender and some of the intermediaries in the transmission are faulty (this is also known as the problem of *Byzantine Agreement* [34]). It is well known [35] that interactive consistency cannot be achieved in the presence of a single arbitrary fault with less than two rounds of information exchange (one to disseminate the values, and one to cross-check), yet TTA sends each message in only a single broadcast. How can we reconcile this practice with theory? I suggest in [55] that the interaction of message broadcasts with the group membership algorithm (which can be seen as a continuously interleaving two-round algorithm) in TTA achieves a "Draconian consensus" in which agreement is enforced by removal of any members that disagree. It would be interesting to subject this idea to formal examination, and to construct an integrated formal treatment for application-level fault tolerance in TTA similar to those previously developed for classical state machine replication [13, 48].

The final top-level property is the most important for safety-critical applications; it is called *partitioning* and it refers to the requirement that faults in one component of TTA, or in one application supported by TTA, must not propagate to other components and applications, and must not affect the operation of nonfaulty components and applications, other than through loss of the services provided by the failed elements. It is quite easy to develop a formal statement of partitioning—but only in the absence of the qualification introduced in the final clause of the previous sentence (see [51] for an extended discussion of this topic). In the absence of communication, partitioning is equivalent to isolation and this property has a long history of formal analysis in the security community [47] and has been adapted to include the real-time attributes that are important in embedded systems [79]. In essence, formal statements of isolation state that the behavior perceived by one component is entirely unchanged by the presence or absence of other components. When communication between components is allowed, this simple statement no longer suffices, for if X_1 supplies input to X_2, then absence of X_1 certainly changes the behavior perceived by X_2. What we want to say is that the *only* change perceived by X_2 is that due to the faulty or missing data supplied by X_1 (i.e., X_1 must not be able to interfere with X_2's communication with other components, nor write directly into its memory, and so on). To my knowledge, there is no fully satisfactory formal statement of this interpretation of partitioning.

It is clear that properties of the TTA algorithms and architecture are crucial to partitioning (e.g., clock synchronization, the global schedule, existence of guardians, the single-fault assumption, and transmission window timing are all needed to stop a faulty node violating partitioning by babbling on the bus), and there are strong informal arguments (backed by experiment) that these properties are sufficient [55], but to my knowledge there is as yet no comprehensive formal treatment of this argument.

7 Conclusion

TTA provides several challenging formal verification problems. Those who wish to develop or benchmark new techniques or tools can find good test cases among the algorithms and requirements of TTA. However, I believe that the most interesting and rewarding problems are those that concern the interactions of several algorithms, and it is here that new methods of compositional analysis and verification are most urgently needed. Examples include the interaction between the group membership and clique avoidance algorithms and their joint behavior under various fault hypotheses, the mutual interdependence of clock synchronization and group membership, and the top-level properties that emerge from the collective interaction of all the algorithms and architectural attributes of TTA. Progress on these fronts will not only advance the techniques and tools of formal methods, but will strengthen and deepen ties between the formal methods and embedded systems communities, and make a valuable contribution to assurance for the safety-critical systems that are increasingly part of our daily lives.

Acknowledgments

Günther Bauer of TU Vienna provided helpful comments and corrections for a previous version of this paper.

References

Papers on formal methods and automated verification by SRI authors can generally be located by visiting home pages or doing a search from `http://www.csl.sri.com/programs/formalmethods`.

[1] James H. Anderson. Lamport on mutual exclusion: 27 years of planting seeds. In *20th ACM Symposium on Principles of Distributed Computing*, pages 3–12, Association for Computing Machinery, Newport, RI, August 2001.

[2] Anish Arora and Sandeep S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In *18th International Conference on Distributed Computing Systems*, pages 436–443, IEEE Computer Society, Amsterdam, The Netherlands, 1998.

[3] Günther Bauer and Michael Paulitsch. An investigation of membership and clique avoidance in TTP/C. In *19th Symposium on Reliable Distributed Systems*, Nuremberg, Germany, October 2000.

[4] Kai Baukus, Saddek Bensalem, Yassine Lakhnech, and Karsten Stahl. Abstracting WS1S systems to verify parameterized networks. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, pages 188–203, Berlin, Germany, March 2000.

[5] Kai Baukus, Yassine Lakhnech, and Karsten Stahl. Verifying universal properties of parameterized networks. In Matthai Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Volume 1926 of Springer-Verlag *Lecture Notes in Computer Science*, pages 291–303, Pune, India, September 2000.

[6] Saddek Bensalem, Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu, and Yassine Lakhnech. A transformational approach for generating non-linear invariants. In Jens Palsberg, editor, *Seventh International Static Analysis Symposium (SAS'00)*, Volume 1824 of Springer-Verlag *Lecture Notes in Computer Science*, pages 58–74, Santa Barbara CA, June 2000.

[7] Saddek Bensalem and Yassine Lakhnech. Automatic generation of invariants. *Formal Methods in Systems Design*, 15(1):75–92, July 1999.

[8] Ahmed Bouajjani and Agathe Merceron. Parametric verification of a group membership algorithm. In Werner Damm and Ernst-Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Volume 2469 of Springer-Verlag *Lecture Notes in Computer Science*, pages 311–330, Oldenburg, Germany, November 2002.

[9] Tushar D. Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In *Fifteenth ACM Symposium on Principles of Distributed Computing*, pages 322–330, Association for Computing Machinery, Philadelphia, PA, May 1996.

[10] Ian G. Clark. *A Unified Approach to the Study of Asynchronous Communication Mechanisms in Real Time Systems*. PhD thesis, King's College, London University, May 2000.

[11] James Corbett, Matthew Dwyer, John Hatcliff, Corina Pasareanu, Robby, Shawn Laubach, and Hongjun Zheng. Bandera: Extracting finite-state models from Java source code. In *22nd International Conference on Software Engineering*, pages 439–448, IEEE Computer Society, Limerick, Ireland, June 2000.

[12] S. J. Creese and A. W. Roscoe. TTP: A case study in combining induction and data independence. Technical Report PRG-TR-1-99, Oxford University Computing Laboratory, Oxford, England, 1999.

[13] Ben L. Di Vito and Ricky W. Butler. Formal techniques for synchronized fault-tolerant systems. In C. E. Landwehr, B. Randell, and L. Simoncini, editors, *Dependable Computing for Critical Applications—3*. Volume 8 of Springer-Verlag, Vienna, Austria *Dependable Computing and Fault-Tolerant Systems*, pages 163–188, September 1992.

[14] Jacob Elgaard, Nils Klarlund, and Anders Möller. Mona 1.x: New techniques for WS1S and WS2S. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer-Aided Verification, CAV '98*, Volume 1427 of Springer-Verlag *Lecture Notes in Computer Science*, pages 516–520, Vancouver, Canada, June 1998.

[15] E. A. Emerson and A. P. Sistla, editors. *Computer-Aided Verification, CAV '2000*, Volume 1855 of Springer-Verlag *Lecture Notes in Computer Science*, Chicago, IL, July 2000.

[16] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonization and Solving. In G. Berry, H. Comon, and A. Finkel, editors, *Computer-Aided Verification, CAV '2001*, Volume 2102 of Springer-Verlag *Lecture Notes in Computer Science*, pages 246–249, Paris, France, July 2001.

[17] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[18] N. Henderson and S. E. Paynter. The formal classification and verification of Simpson's 4-slot asynchronous communication mechanism. In Peter Lindsay, editor, *FME 2002: Formal Methods–Getting IT Right*, pages 350–369, Copenhagen, Denmark, July 2002.

[19] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: a time-triggered language for embedded programming. In Henzinger and Kirsch [20], pages 166–184.

[20] Tom Henzinger and Christoph Kirsch, editors. *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, Volume 2211 of Springer-Verlag *Lecture Notes in Computer Science*, Lake Tahoe, CA, October 2001.

[21] Wim H. Hesselink. An assertional criterion for atomicity. *Acta Informatica*, 28(5):343–366, 2002.

[22] C. B. Jones. Tentative steps toward a development method for interfering programs. *ACM TOPLAS*, 5(4):596–619, 1983.

[23] Shmuel Katz, Pat Lincoln, and John Rushby. Low-overhead time-triggered group membership. In Marios Mavronicolas and Philippas Tsigas, editors, *11th International Workshop on Distributed Algorithms (WDAG '97)*, Volume 1320 of Springer-Verlag *Lecture Notes in Computer Science*, pages 155–169, Saarbrücken Germany, September 1997.

[24] Herman Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. In *6th IEEE Workshop on Future Trends in Distributed Computing*, pages 310–315, IEEE Computer Society, Tunis, Tunisia, October 1997.

[25] Hermann Kopetz. *Real-Time Systems: Design Princples for Distributed Embedded Applications*. The Kluwer International Series in Engineering and Computer Science. Kluwer, Dordrecht, The Netherlands, 1997.

[26] Hermann Kopetz. The time-triggered model of computation. In *Real Time Systems Symposium*, IEEE Computer Society, Madrid, Spain, December 1998.

[27] Hermann Kopetz. Elementary versus composite interfaces in distributed real-time systems. In *The Fourth International Symposium on Autonomous Decentralized Systems*, IEEE Computer Society, Tokyo, Japan, March 1999.

[28] Hermann Kopetz and Günter Grünsteidl. TTP—a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.

[29] Hermann Kopetz and Johannes Reisinger. The non-blocking write protocol NBW: A solution to a real-time synchronization problem. In *Real Time Systems Symposium*, pages 131–137, IEEE Computer Society, Raleigh-Durham, NC, December 1993.

[30] Sandeep Kulkarni, John Rushby, and N. Shankar. A case study in component-based mechanical verification of fault-tolerant programs. In *ICDCS Workshop on Self-Stabilizing Systems*, pages 33–40, IEEE Computer Society, Austin, TX, June 1999.

[31] Sandeep S. Kulkarni. *Component-Based Design of Fault Tolerance*. PhD thesis, The Ohio State University, Columbus, OH, 1999.

[32] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[33] Leslie Lamport. Concurrent reading and writing. *Association for Computing Machinery*, 20(11):806–811, November 1977.

[34] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[35] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco, CA, 1996.

[36] Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems*, 8(4):284–304, November 1990.

[37] K. L. McMillan. Circular compositional reasoning about liveness. In Laurence Pierre and Thomas Kropf, editors, *Advances in Hardware Design and Verification: IFIP WG10.5 International Conference on Correct Hardware Design and Verification Methods (CHARME '99)*, Volume 1703 of Springer-Verlag *Lecture Notes in Computer Science*, pages 342–345, Bad Herrenalb, Germany, September 1999.

[38] Paul S. Miner. Verification of fault-tolerant clock synchronization systems. NASA Technical Paper 3349, NASA Langley Research Center, Hampton, VA, November 1993.

[39] Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, July 1981.

[40] Kedar S. Namjoshi and Richard J. Trefler. On the completeness of compositional reasoning. In Emerson and Sistla [15], pages 139–153.

[41] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[42] Michael Paulitsch and Wilfried Steiner. The transition from asynchronous to synchronous system operation: An approach for distributed fault-tolerant systems. In *The 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pages 329–336, IEEE Computer Society, Vienna, Austria, July 2002.

[43] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[44] Holger Pfeifer. Formal verification of the TTA group membership algorithm. In Tommaso Bolognesi and Diego Latella, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification FORTE XIII/PSTV XX 2000*, pages 3–18, Pisa, Italy, October 2000.

[45] Holger Pfeifer, Detlef Schwier, and Friedrich W. von Henke. Formal verification for time-triggered clock synchronization. In Weinstock and Rushby [77], pages 207–226.

[46] Stefan Poledna. *Fault-Tolerant Systems: The Problem of Replica Determinism*. The Kluwer International Series in Engineering and Computer Science. Kluwer, Dordrecht, The Netherlands, 1996.

[47] John Rushby. The design and verification of secure systems. In *Eighth ACM Symposium on Operating System Principles*, pages 12–21, Asilomar, CA, December 1981. (ACM *Operating Systems Review*, Vol. 15, No. 5).

[48] John Rushby. A fault-masking and transient-recovery model for digital flight-control systems. In Jan Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Kluwer International Series in Engineering and Computer Science, chapter 5, pages 109–136. Kluwer, Boston, Dordecht, London, 1993.

[49] John Rushby. A formally verified algorithm for clock synchronization under a hybrid fault model. In *Thirteenth ACM Symposium on Principles of Distributed Computing*, pages 304–313, Association for Computing Machinery, Los Angeles, CA, August 1994. Also available as NASA Contractor Report 198289.

[50] John Rushby. Automated deduction and formal methods. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, Volume 1102 of Springer-Verlag

Lecture Notes in Computer Science, pages 169–183, New Brunswick, NJ, July/August 1996.

[51] John Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Available at `http://www.csl.sri.com/~rushby/abstracts/partitioning`, and `http://techreports.larc.nasa.gov/ltrs/PDF/1999/cr/NASA-99-cr209347.pdf`; also issued by the FAA.

[52] John Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. *IEEE Transactions on Software Engineering*, 25(5):651–660, September/October 1999.

[53] John Rushby. Verification diagrams revisited: Disjunctive invariants for easy verification. In Emerson and Sistla [15], pages 508–520.

[54] John Rushby. Bus architectures for safety-critical embedded systems. In Henzinger and Kirsch [20], pages 306–323.

[55] John Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, September 2001. Available at `http://www.csl.sri.com/~rushby/abstracts/buscompare`.

[56] John Rushby. Formal verification of McMillan's compositional assume-guarantee rule. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, September 2001.

[57] John Rushby. Formal verification of transmission window timing for the time-triggered architecture. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, March 2001.

[58] John Rushby. Formal verification of Marzullo's sensor fusion interval. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, January 2002.

[59] John Rushby. Model checking Simpson's four-slot fully asynchronous communication mechanism. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, July 2002.

[60] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, January 1993.

[61] John Rushby, Friedrich von Henke, and Sam Owre. An introduction to formal specification and verification using EHDM. Technical Report SRI-CSL-91-2, Computer Science Laboratory, SRI International, Menlo Park, CA, February 1991.

[62] Hassen Saïdi and Susanne Graf. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer-Aided Verification, CAV '97*, Volume 1254 of Springer-Verlag *Lecture Notes in Computer Science*, pages 72–83, Haifa, Israel, June 1997.

[63] Hassen Saïdi and N. Shankar. Abstract and model check while you prove. In Nicolas Halbwachs and Doron Peled, editors, *Computer-Aided Verification, CAV '99*, Volume 1633 of Springer-Verlag *Lecture Notes in Computer Science*, pages 443–454, Trento, Italy, July 1999.

[64] Ulrich Schmid. How to model link failures: A perception-based fault model. In *The International Conference on Dependable Systems and Networks*, pages 57–66, IEEE Computer Society, Goteborg, Sweden, July 2001.

[65] Ulrich Schmid and Klaus Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing*, 14(2):101–111, May 2001.

[66] Fred B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report 87-859, Department of Computer Science, Cornell University, Ithaca, NY, August 1987.

[67] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[68] Marco Schneider. Self stabilization. *ACM Computing Surveys*, 25(1):45–67, March 1993.

[69] D. Schwier and F. von Henke. Mechanical verification of clock synchronization algorithms. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Volume 1486 of Springer-Verlag *Lecture Notes in Computer Science*, pages 262–271, Lyngby, Denmark, September 1998.

[70] Natarajan Shankar. Mechanical verification of a generalized protocol for Byzantine fault-tolerant clock synchronization. In J. Vytopil, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Volume 571 of Springer-Verlag *Lecture Notes in Computer Science*, pages 217–236, Nijmegen, The Netherlands, January 1992.

[71] Natarajan Shankar. Combining theorem proving and model checking through symbolic analysis. In *CONCUR 2000: Concurrency Theory*, pages 1–16, State College, PA, August 2000. Available at `ftp://ftp.csl.sri.com/pub/users/shankar/concur2000.ps.gz`.

[72] H. R. Simpson. Four-slot fully asynchronous communication mechanism. *IEE Proceedings, Part E: Computers and Digital Techniques*, 137(1):17–30, January 1990.

[73] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[74] Philip Thambidurai and You-Keun Park. Interactive consistency with multiple failure modes. In *7th Symposium on Reliable Distributed Systems*, pages 93–100, IEEE Computer Society, Columbus, OH, October 1988.

[75] *Specification of the TTP/C Protocol (version 0.6p0504)*. Time-Triggered Technology TTTech Computertechnik AG, Vienna, Austria, May 2001.

[76] Ashish Tiwari, Harald Rueß, Hassen Saïdi, and N. Shankar. A technique for invariant generation. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 7th International Conference, TACAS 2001*, Volume 2031 of Springer-Verlag *Lecture Notes in Computer Science*, pages 113–127, Genova, Italy, April 2001.

[77] Charles B. Weinstock and John Rushby, editors. *Dependable Computing for Critical Applications—7*, Volume 12 of IEEE Computer Society *Dependable Computing and Fault Tolerant Systems*, San Jose, CA, January 1999.

[78] J. Lundelius Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, April 1988.

[79] Matthew M. Wilding, David S. Hardin, and David A. Greve. Invariant performance: A statement of task isolation useful for embedded application integration. In Weinstock and Rushby [77], pages 287–300.